Appendix A: Introduction to the ZSL Notation

This is a very brief introduction to Z and ZSL. Z is a formal notation for specifying software systems, based on first order logic and set theory. It uses certain graphical constructs as part of the notation. For this reason, ZSL was invented: it is a purely textual form of Z. ZSL does not add anything to Z beyond an ASCII interface.

Given Sets

Sometimes it is not necessary to give a detailed specification of items that we are trying to model. In that case, the items can be captured by given sets.

[Name, BasicUnit, Book, Human]

Z comes with some built-in given sets, namely the integers, natural numbers, and whole numbers.

[Z, N, N1]
% integers, natural numbers, whole numbers respectively

To represent set membership:

3 in N
-3 not in N1

Types

We can use given sets in declarations of variables to act as the type of the variable. Types can also be drawn from other Z constructs (e.g. schemas).
nameofsystem : Name
population : N

Building Sets

We sometimes want to build sets out of other types.

names : P Name
% a set of names

nameandages : P (Name cross Z)
% a set of binary tuples - name with integers

nameandages2 : Name <-> Z
% the same as the previous: it is a relation

humantooage : Name --> Z
% a relation which is also a function

Predicates (Constraints)

First order predicates are supported in Z.

forall n : N1 | n in setofbignumbers @
  n > 1000
% some set of big numbers all have numbers greater than 1000

exists min : N1 | min in setofbignumbers @
  forall n : setofbignumbers @ min <= n
% the set of big numbers has a minimum

Declaring Variables

When we want to declare a variable, we must specify its type and some optional constraints (also known as axioms in this case).

global
  population : Z
axiom
  population > 100
  population < 6000000000
end axiom

The list of axioms is an implicit conjunction.
Schemas

The basic unit for capturing state in Z is the schema. Schemas have two parts: a signature which captures attributes, and a property which captures constraints about the attributes in the signature.

```
schema Car
  type : CarType
  passengers : P Name
  wheels : N1
  where
    # passengers < 6
    % the cardinality of the set passengers is less than 6
    wheels <= 4
end schema
```

The list of constraints in the property are implicitly part of a conjunction.

Schema Operations

Schemas can be used to describe operations which affect the state of the attributes, and which can also take inputs and outputs.

```
schema AddPassenger
  Delta Car
  % the Car schema is changing
  newp? : Passenger
  % an input passenger
  currentpassengers! : N1
  % output number of current passengers after the operation
  where
    newp? notin passengers
    % new passenger not in the set of current passengers
    # passengers < 5
    % there is still room in the car
    passengers' = passengers union {newp?}
    % the postoperation set of passengers is the preoperation union with the set of (a single) passenger
end schema
```
In general, inputs end with a ‘?’; outputs with a ‘!’; and postoperation state attributes with an apostrophe. Preoperation attributes do not end with anything special.
Appendix B: Abd-Allah’s Z Model

This is the complete AAA model written in Z by Abd-Allah for his dissertation [Abd-Allah 1996].

```z
generic [X]
    uconnected : (X & X) <-> (X <-> X)
    % true iff a pair of nodes are connected by a set of edges in an undirected graph
    % (where the edges are given as a set of binary tuples of nodes)
    where
    forall a, b : X; edges : X <-> X @
        uconnected (a,b) = edges <=>
        ((exists c : X @
            ((a,c) in edges or (c,a) in edges) and
            uconnected (c, b) = edges)
        % an edge exists from node a to some other node c, and nodes b and c are
        connected (recursive definition)
        or
        (exists a : X @ a=b))
        % trivial case: each node is connected to itself
    end generic

generic [X]
    dconnected : (X & X) <-> (X <-> X)
    % true iff a pair of nodes are connected by a set of edges in a directed graph
    % (where the edges are given as a set of binary tuples of nodes)
    where
    forall a, b : X; edges : X <-> X @
        dconnected (a,b) = edges <=>
        ((exists c : X @
            (a,c) in edges and dconnected (c, b) = edges)
        % an edge exists from node a to some other node c, and nodes b and c are
        connected (recursive definition)
        or
        (exists a : X @ a=b))
        % trivial case: each node is connected to itself
    end generic

generic [X]
    cugraph : P X <-> (X <-> X)
    % true iff an undirected graph is connected (where the graph is represented as a
    % set of nodes and a set of edges)
    where
    forall nodes : P X; edges : X <-> X @
        cugraph nodes = edges <=>
        (#nodes > 1 and dom edges = nodes and ran edges = nodes
        and (forall a, b : nodes @ uconnected (a,b) = edges))
```
% all nodes are connected to each other
or
(#nodes = 0 or #nodes = 1))
% trivial case: a graph of zero or one node
end generic

generic [X]
cdgraph : (X & P X) <-> (X <-> X)
% true iff a directed graph is connected with respect to a given `root' node (where the graph is represented as a set of nodes and a set of edges)
where
forall root : X; nodes : P X; edges : X <-> X | root in nodes @
cdgraph (root, nodes) = edges <=>
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and (forall a : nodes @ dconnected (root, a) = edges))
% all nodes are connected to the root
or
(#nodes = 0 or #nodes = 1))
% trivial case: a graph of zero or one node
end generic

generic [X]
utree : P X <-> (X <-> X)
% true iff an undirected graph is a tree (where the graph is represented as a set of nodes and a set of edges)
where
forall nodes : P X; edges : X <-> X @
utree nodes = edges <=>
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and #edges = #nodes - 1
and (forall a, b : nodes @ uconnected (a,b) = edges))
% the number of edges is one less than the number of nodes, and all nodes are connected to each other
or
(#nodes = 0 or #nodes = 1))
% trivial case: a tree of zero or one node
end generic

generic [X]
dtree : (X & P X) <-> (X <-> X)
% true iff a directed graph is a tree with respect to a given `root' node (where the graph is represented as a set of nodes and a set of edges)
where
dtree (root, nodes) = edges <=>
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and #edges = #nodes - 1
and (forall a : nodes @ dconnected (root, a) = edges))
% the number of edges is one less than the number of nodes, and all nodes are connected to the root
or
trivial case: a tree of zero or one node
end generic

generic [X]
ustar : P X <-> (X <-> X)
% true iff an undirected graph is a star
where
  forall nodes : P X; edges : X <-> X @
  ustar nodes = edges <==>
  ((utree nodes = edges) and
   (exists center : X @
    #nodes > 1 =>
    (forall a,b : X | (a,b) in edges @ a = center or b = center))))
% the graph is a tree, and one node is found in all the edges
end generic

generic [X]
dstar : (X & P X) <-> (X <-> X)
% true iff a directed graph is a star
where
  forall center : X; nodes : P X; edges : X <-> X | center in nodes @
  dstar (center, nodes) = edges <==>
  ((dtree (center, nodes) = edges) and
   #nodes > 1 =>
   (forall a,b : X | (a,b) in edges @ a = center))
% the graph is a tree, and one node is found in all the edges (the root)
end generic

generic [X]
uring : P X <-> (X <-> X)
% true iff an undirected graph is a ring
where
  forall nodes : P X; edges : X <-> X @
  uring nodes = edges <==>
  ((#nodes > 2 and dom edges = nodes and ran edges = nodes and
   #edges = #nodes and
   (forall a : nodes @
    #{(x, y : X | (x,y) in edges and (x=a or y=a)) = 2})
    % the number of edges is equal to the number of nodes, and each node is
    found in 2 edges
  or
  (#nodes = 0 or #nodes = 1 or #nodes = 2))
% trivial case: a ring of zero, one, or two nodes
end generic

generic [X]
ufullyconnectedgraph : P X <-> (X <-> X)
% true iff an undirected graph is fully connected
where
  forall nodes : P X; edges : X <-> X @

ufullyconnectedgraph nodes = edges <=>
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and #edges = (#nodes * (#nodes - 1)) div 2 and
(forall a,b : nodes @
(a,b) in edges or (b,a) in edges))
% the number of edges is equal to the number of nodes multiplied by the same
minus one, all divided by two, and an edge exists between all pairs of
nodes
or
(#nodes = 0 or #nodes = 1))
% trivial case: a fully connected graph of zero or one node
end generic

[CpuType, DistributedCommunicationMechanismType, OperatingSystemType, GraphicalUserInterfaceType, ImplementationLanguageType]
% sets of platform options

global
sparc, i80x86, m680x0, powerpc : CpuType;
rpc, messages, sharedmemory : DistributedCommunicationMechanismType;
sunos, solaris, aix, windows3, windows95,
dos, os2, system7, vms : OperatingSystemType;
xwindowsgui, windows3gui, windows95gui,
system7gui, os2gui : GraphicalUserInterfaceType;
c, cplusplus, ada, pascal, prolog, lisp, fortran,
cobol, smalltalk, basic : ImplementationLanguageType;
end global
% partial listing of the possible platform options

[VendorType, VersionType]

schema PlatformOption [X]
  type : X;
  vendor : VendorType;
  version : VersionType;
end schema

% schema for platform options

schema Platform
  cpumodel : PlatformOption[CpuModelType];
discommmech :
    PlatformOption[DistributedCommunicationMechanismType];
language : PlatformOption[ImplementationLanguageType];
os : PlatformOption[OperatingSystemType];
gui : PlatformOption[GraphicalUserInterfaceType];
end schema

%% inrel PlatformCompatible

global
  _ PlatformCompatible _ : Platform <-> Platform;
% determine if two platforms are compatible with each other
axiom
  forall p1, p2 : Platform @
  p1.cpumodel = p2.cpumodel and
  p1.discommmech = p2.discommmech and
  p1.language = p2.language and
  p1.os = p2.os and
  p1.gui = p2.gui;
end axiom

schema Node
  platform : Platform;
end schema

Percent == {n : N | n <= 100}
% percent type

schema Resources
  cpuload : Percent;
  % cpu load (percent of total load)
  sloc : N1;
  % source lines of code (lines)
  datasize : N1;
  % data size (bytes)
  codesize : N1;
  % code size (bytes)
  mainmemory : N1;
  % main memory (bytes)
  networkbandwidth : N1;
  % network bandwidth (bytes per second)
end schema

% % inrel ResourceCompatible

global
  NoResources : Resources;
  % ‘zero’ for resources schema
  addResources : (P Resources) <-> Resources;
  % adds a set of resources together
  _ ResourceCompatible _ : Resources <-> Resources;
  % determine if one set of resources is compatible with another
axiom
  NoResources.sloc = 0;
  NoResources.datasize = 0;
  NoResources.codesize = 0;
  NoResources.cpuload = 0;
  NoResources.mainmemory = 0;
  NoResources.networkbandwidth = 0;
  forall inputset : P Resources; r1, r2, r3 : Resources @
    (inputset = {} => addResources inputset = NoResources) and
    (inputset = {r1} => addResources inputset = r1) and
(((r1, r2) \subseteq \text{inputset} \text{ and } r3 \notin \text{inputset}) \Rightarrow 
\text{addResources}(\text{inputset}) = \text{addResources}((\text{inputset} \setminus \{r1, r2\}) 
\cup \{r3\}) \text{ and} 
\begin{align*}
    r3.\text{sloc} &= r1.\text{sloc} + r2.\text{sloc} \\
    r3.\text{datasize} &= r1.\text{datasize} + r2.\text{datasize} \\
    r3.\text{codesize} &= r1.\text{codesize} + r2.\text{codesize} \\
    r3.\text{cpuload} &= r1.\text{cpuload} + r2.\text{cpuload} \\
    r3.\text{mainmemory} &= r1.\text{mainmemory} + r2.\text{mainmemory} \\
    r3.\text{networkbandwidth} &= r1.\text{networkbandwidth} + 
\end{align*}
\begin{align*}
    r2.\text{networkbandwidth));}
\end{align*}
\begin{align*}
    \forall r_{\text{actual}}, r_{\text{limits}} : \text{Resources} @ \\
    \quad r_{\text{actual}} \text{ResourceCompatible} r_{\text{limits}} \iff \\
    \quad r_{\text{actual}}.\text{sloc} \leq r_{\text{limits}}.\text{sloc} \text{ and} \\
    \quad r_{\text{actual}}.\text{datasize} \leq r_{\text{limits}}.\text{datasize} \text{ and} \\
    \quad r_{\text{actual}}.\text{codesize} \leq r_{\text{limits}}.\text{codesize} \text{ and} \\
    \quad r_{\text{actual}}.\text{cpuload} \leq r_{\text{limits}}.\text{cpuload} \text{ and} \\
    \quad r_{\text{actual}}.\text{mainmemory} \leq r_{\text{limits}}.\text{mainmemory} \text{ and} \\
    \quad r_{\text{actual}}.\text{networkbandwidth} \leq r_{\text{limits}}.\text{networkbandwidth}; 
\end{align*}
\end{axiom}

[Name]
% given set of names for components, connectors, etc.

\textbf{Boolean} ::= TRUE | FALSE
% boolean type

[DataType]
% given set for primitive data types
% declare a few common primitive data types (can be extended)
global
    \text{character}, \text{integer}, \text{float}, \text{array}, \text{structure}, \text{string} : \text{DataType};
end global

\textbf{IoType} ::= \text{io\_in} | \text{io\_out} | \text{io\_inout}
% for indicating the I/O type of a port
% schema for high-level (data) port
schema Port
\begin{align*}
    \text{name} &: \text{Name}; \\
    \text{iotype} &: \text{IoType}; \\
    \quad \% \text{handles input, output, or both} \\
    \text{dataformat} &: \text{seq} \text{DataType}; \\
    \quad \% \text{format of the expected data} \\
    \text{rate} &: \text{N1}; \\
    \quad \% \text{maximum input and/or output rate} \\
    \text{buffersize} &: \text{N}; \\
    \quad \% \text{buffersize - 0 if not buffered} \\
    \text{streaming} &: \text{Boolean}; \\
    \quad \% \text{true if the transfer streams data incrementally} \\
    \text{blocking} &: \text{Boolean}; \\
\end{align*}
% true if the transfer blocks on output (not applicable to io_in ports)  
maximumconnections : N1;  
% maximum connections allowed  
end schema

% declare two common ports that are not attached to any particular control component  
or object  
global  
  stdin, stdout : Port;  
axiom  
  stdin.iotype = io_in;  
  stdout.iotype = io_out;  
end axiom

% schema for high-level data component  
schema DataComponent  
  name : Name;  
  type : seq DataType;  
  % type of the data determined by internal structure  
  nodes : P Node;  
  % nodes it resides on  
  resources : Resources;  
  % resource usage  
end schema

% schema for high-level control component  
schema ControlComponent  
  name : Name;  
  ports : P Port;  
  arguments : seq DataType;  
  % expected arguments if called or spawned  
  returnvalues : seq DataType;  
  % type of returned values when returning from a call or spawn  
  locals : P DataComponent;  
  % local variables  
  localobjects : P Name;  
  % local objects (given only by name - unique)  
  method : Boolean;  
  % indicate whether it is an object method or not  
  nodes : P Node;  
  % nodes it resides on  
  resources : Resources;  
  % resource usage  
  platform : Platform;  
  % expected platform  
end schema

[Class]  
% given set of classes  

% schema for high-level object
schema Object
  name : Name;
  class : Class;
  % class the object belongs to
  ports : P Port;
  % object ports
  data : P DataComponent;
  % private data of the object
  methods : P ControlComponent;
  % control components which act as the object’s methods
  nodes : P Node;
  % nodes it resides on
  resources : Resources;
  % resource usage
  platform : Platform;
  % expected platform
where
  forall c : methods @
    c.method = TRUE and
    c.nodes subseteq nodes and
    c.platform PlatformCompatible platform;
  % all the methods reside on object nodes, and are compatible with the object’s
  platform
  addResources {r : Resources | exists m : methods @ r = m.resources}
    ResourceCompatible resources;
  % combined method resources are compatible with object resources
end schema

% ensure no methods exist without a parent object
axiom
  forall c : ControlComponent @
    c.method = TRUE <=> (exists o : Object @ c in o.methods);
end axiom

% two functions for retrieving objects and methods
global
  r_getallobjects : P Object +-> P Object;
  % recursively gets all objects accessible via input objects’ methods
  r_getallmethods : P Object +-> P ControlComponent;
  % recursively gets all methods accessible via input objects’ methods
axiom
  forall inobjects, outobjects : P Object @
    r_getallobjects(inobjects) = outobjects <=>
      outobjects = inobjects setunion r_getallobjects({o : Object |
        (exists inob : inobjects; m : ControlComponent |
          m in inob.methods @ o.name in m.localobjects)});
  forall inobjects : P Object; methods : P ControlComponent @
    r_getallmethods(inobjects) = methods <=>
      methods = {m : ControlComponent |
        (exists inob : inobjects @ m in inob.methods)}
      setunion r_getallmethods({o : Object |
(exists inob : inobjects; m : ControlComponent |
   m in inob.methods @ o.name in m.localobjects)));

end axiom

COsubtype ::= ctype | otype
% a useful type for indicating whether a property applies to control components or
   objects

Directionality ::= forward | reverse | twoway
% a useful type for data connectors

% schema for high-level data connector
schema DataConnector
   name : Name;
   p1, p2 : Port;
   % connected ports
   c1, c2 : ControlComponent;
   % connected control components
   o1, o2 : Object;
   % connected objects (if the control components are methods or if the particular
   spec hasn't been refined to include specific methods)
   cc_or_obj : COsubtype;
   % indicate if an object or a control component is being connected
   dataformat : seq DataType;
   % format of the data expected to be transferred
   rate : N1;
   % maximum transfer rate
   directionality : Directionality;
   % indicate direction of communications
   buffersize : N;
   % buffersize - 0 if not buffered
   reliable : Boolean;
   % true if each transfer is reliably sent
   nodes : P Node;
   % nodes it resides on
   resources : Resources;
   % resource usage
   platform : Platform;
   % expected platform

where
   cc_or_obj = ctype => p1 in c1.ports and p2 in c2.ports;
   cc_or_obj = otype => p1 in o1.ports and p2 in o2.ports;
   % the connected ports belong to the specified components or objects
   c1.method = TRUE => c1 in o1.methods;
   c2.method = TRUE => c2 in o2.methods;
   % if either component is a method, it belongs to the respective object
   ((p1.iotype = io_in and p2.iotype = io_out) or
    (p1.iotype = io_in and p2.iotype = io_inout) or
    (p1.iotype = io_inout and p2.iotype = io_inout) or
    (p2.iotype = io_in and p1.iotype = io_out) or
    (p2.iotype = io_in and p1.iotype = io_inout));
% allowable port combinations: in - out, inout - inout, in - inout
{n : Node | n in c1.nodes or n in c2.nodes or n in o1.nodes
or n in o2.nodes} subseteq nodes;
% the connector nodes form a superset of the component/object nodes
end schema

% schema for high-level control connector
schema ControlConnector
  name : Name;
  c1, c2 : ControlComponent;
% connected control components
  o1, o2 : Object;
% connected objects (if the control components are methods)
  arguments : seq DataType;
% arguments passed in during a call or spawn (if any)
  returnvalues : seq DataType;
% values when returning from a call or spawn (if any)
  rate : N1;
% maximum transfer rate
  queuesize : N;
% queuesize - 0 if connector does not queue requests
  blocking : Boolean;
% true if the transfer blocks waiting for a response
  reliable : Boolean;
% true if each transfer is reliably sent
  nodes : P Node;
% nodes it resides on
  resources : Resources;
% resource usage
  platform : Platform;
% expected platform
where
{n : Node | n in c1.nodes or n in c2.nodes} subseteq nodes;
% the connector nodes form a superset of the component nodes
c1.method = TRUE <= c1 in o1.methods and
c2.method = TRUE <= c2 in o2.methods;
% if either component is a method, it belongs to the respective object
  arguments = c2.arguments;
  returnvalues = c2.returnvalues;
end schema

Call == {c : ControlConnector | c.blocking = TRUE}

Spawn == {c : ControlConnector | c.blocking = FALSE}

axiom
  forall c : Spawn @ c.c1.method = FALSE and c.c2.method = FALSE;
% impose artificial constraint: no spawns to/from object methods
end axiom

% schema for high-level trigger
schema Trigger
  name : Name;
  subtype : Csubtype;
    % indicate if an object or a control component is being connected
controlcomponent : ControlComponent;
    % the control component which receives the data message
object : Object;
    % the object which receives the data message
inmessage : DataComponent;
    % the input message
inports : P Port;
    % the input port which the message come in over
outports : P Port;
    % the output ports for any outgoing messages
outmessageset : P DataComponent;
    % the set of triggered outgoing messages
outcalls : P ControlComponent;
    % the set of triggered calls
outspawns : P ControlComponent;
    % the set of triggered spawns

where
  forall s : Port | s in inports @
    s.iotype = io_in or s.iotype = io_inout;
    % in ports accept input
  forall s : Port | s in outports @
    s.iotype = io_out or s.iotype = io_inout;
    % out ports allow output

subtype = ctype =>
  inports setunion outports subseteq controlcomponent.ports and
  controlcomponent.method = FALSE;
    % if the trigger is for a control component, the ports are indeed found in the specified control component, and it is not a method

subtype = otype =>
  inports setunion outports subseteq object.ports and
  outcalls subseteq object.methods and
  outspawns = {};
    % if the trigger is for an object, the ports are indeed found in the specified object. Calls are to object methods, and no spawns are allowed of object methods (artificial constraint).

end schema

Layer == P ControlComponent & P ControlComponent
  % Layer is a Z abbreviation definition for two sets of control components (top half and bottom half of layer)

Layers == P ControlComponent <-> P ControlComponent
  % An abbreviation definition for a set of layers

Thread == P ControlComponent
  % A thread is a set of control components that are presumably connected by calls (not spawns). "Thread" should only be used in the proper context otherwise use "P
ControlComponent”.

% schema for high-level system

schema System
    name : Name;
    initialcontrolcomponents : P ControlComponent;
    % initially running control components
    initialdataconnectors : P DataConnector;
    % initially running data connectors
    globalobjects : P Object;
    % global objects
    controlcomponents : P ControlComponent;
    % all system control components, excluding object methods
    classes : P Class;
    % allowed system classes
    shareddata : DataComponent +-> (P ControlComponent & P Object);
    % all system shared data
    dataconnectors : P DataConnector;
    % all system data connectors
    calls : P Call;
    % all system blocking control connectors
    spawns : P Spawn;
    % all system non-blocking control connectors
    recognizedmessages : P DataComponent;
    % system recognized messages (events)
    triggers : P Trigger;
    % all system triggers
    call_layers : Layers;
    % layers with respect to calls (top, bottom)
    spawn_layers : Layers;
    % layers with respect to spawns (top, bottom)
    dataconnector_layers : Layers;
    % layers with respect to dataconnectors (top, bottom)
    dataconnectorattributes : DataConnector;
    % the global attributes of the data connectors in the system
    controlconnectorattributes : ControlConnector;
    % the global attributes of the control connectors in the system
    nodes : P Node;
    % available nodes
    resources : Resources;
    % resource usage
    platform : Platform;
    % expected platform

where
    # initialcontrolcomponents + # globalobjects >= 1;
    % there is at least one initially running control component or global object in the system
    initialcontrolcomponents subseteq controlcomponents;
    % all initially running control components belong to declared set of control components
    forall o : Object |
o in r_getallobjects(globalobjects) or
  o in r_getallobjects({o : Object | exists c : controlcomponents; n : Name |
    n in c.localobjects @ n = o.name}) @
  o.class in classes;
% all global/local objects belong to a system class
initialdataconnectors subseteq dataconnectors;
% all initially running dataconnectors belong to declared set of dataconnectors
{c : ControlComponent |
  (exists call : calls @ c = call.c1 or c = call.c2) or
  (exists spawn : spawns @ c = spawn.c1 or c = spawn.c2) or
  (exists dc : dataconnectors @ c = dc.c1 or c = dc.c2)}
subseteq
  controlcomponents setunion r_getallmethods(globalobjects) setunion
  r_getallmethods(r_getallobjects({o : Object |
    exists c : controlcomponents; n : Name |
    n in c.localobjects @ n = o.name}));
% all components connected by calls, spawns, and/or dataconnectors are mem-
% bers of the system control components or of methods of global/local objects
{o : Object |
  (exists dc : dataconnectors | dc.cc_or_obj = otype @
    o = dc.o1 or o = dc.o2)}
subseteq
  r_getallobjects(globalobjects) setunion r_getallobjects({o : Object |
    exists c : controlcomponents; n : Name |
    n in c.localobjects @ n = o.name})
% all objects connected by dataconnectors are global or local objects
{s : Port | (exists dc : dataconnectors @ s = dc.p1 or s = dc.p2)} subseteq
  {s : Port | (exists c : ControlComponent | c in controlcomponents setunion
    r_getallmethods(globalobjects) setunion
    r_getallmethods(r_getallobjects({o : Object |
      exists c : controlcomponents; n : Name |
      n in c.localobjects @ n = o.name})) @ s in c.ports) or
  (exists c : ControlComponent | c in controlcomponents setunion
    r_getallmethods(globalobjects) setunion
    r_getallmethods(r_getallobjects({o : Object |
      exists c : controlcomponents; n : Name |
      n in c.localobjects @ n = o.name})) @ s in c.ports) or
  (exists o : Object | o in r_getallobjects(globalobjects) or
    o in r_getallobjects({o : Object | exists c : controlcomponents; |
    n : Name | n in c.localobjects @ n = o.name}) @ s in c.ports));
% dataconnector ports are subset of controlcomponent, method, and (global/local)
% object ports
forall cset : P ControlComponent; oset : P Object |
  (exists d : DataComponent @ (d, (cset, oset)) in shareddata)
  @ cset subseteq controlcomponents setunion
    r_getallmethods(globalobjects) setunion
    r_getallmethods(r_getallobjects({o : Object |
      exists c : controlcomponents; n : Name |
      n in c.localobjects @ n = o.name})) and
  oset subseteq r_getallobjects(globalobjects) setunin
    r_getallobjects({o : Object | exists c : controlcomponents; n : Name |
      n in c.localobjects @ n = o.name});
% components sharing data are found in controlcomponents/methods and objects
% sharing data are found in global or local objects
% layering constraints currently ignore objects
forall layer : Layer | layer in call_layers or layer in spawn_layers or
layer in dataconnector_layers @
first layer setunion second layer subseteq
controlcomponents setunion
r_getallmethods(globalobjects) setunion
r_getallmethods(r_getallobjects({o : Object |
exists c : controlcomponents; n : Name |
n in c.localobjects @ n = o.name})));

% all layers drawn from the controlcomponents & methods
forall call : ControlConnector | call in calls @
((exists layer : Layer | layer in call_layers @
(call.c1 in first layer and call.c2 in second layer) or
(call.c2 in first layer and call.c1 in second layer))
or
(forall layer : Layer | layer in call_layers @
{call.c1, call.c2} setint second layer = {}));

% calls constrained to between layers or not constrained at all
forall spawn : ControlConnector | spawn in spawns @
((exists layer : Layer | layer in spawn_layers @
(spawn.c1 in first layer and spawn.c2 in second layer) or
(spawn.c2 in first layer and spawn.c1 in second layer))
or
(forall layer : Layer | layer in spawn_layers @
{spawn.c1, spawn.c2} setint second layer = {}));

% spawns constrained to between layers or not constrained at all
forall d : DataConnector | d in dataconnectors @
((exists layer : Layer | layer in dataconnector_layers @
(d.c1 in first layer and d.c2 in second layer) or
(d.c2 in first layer and d.c1 in second layer))
or
(forall layer : Layer | layer in dataconnector_layers @
{d.c1, d.c2} setint second layer = {}));

% dataconnectors constrained to between layers or not constrained at all
forall c : ControlComponent; threads : P Thread |
c in controlcomponents setunion
r_getallmethods(globalobjects) setunion
r_getallmethods(r_getallobjects({o : Object |
exists c : controlcomponents; n : Name |
n in c.localobjects @ n = o.name})) and
threads = {thread : Thread |
(exists topcc : ControlComponent @
(topcc in initialcontrolcomponents or
(exists t : triggers @ topcc in t.outspawns
or topcc in t.outcalls) or
(exists c1 : ControlComponent; spawn : ControlConnector |
spawn in spawns @ spawn.c1 = c1
and spawn.c2 = topcc)) and
thread = {x : ControlComponent |
exists chain : P ControlConnector | chain subseteq calls @
dconnected (topcc, x) = {a,b : ControlComponent |
exists call : ControlConnector |
call in chain @ a = call.c1 and b = call.c2))) @
(exists thread : Thread | thread in threads @ c in thread) and
% every control component and every method is either initial, [trigger] called, or [trigger] spawned
(forall c : ControlConnector | c in calls or c in spawns @
((c.c1.method = TRUE and c.c2.method = TRUE) =>
(c.o2 in globalobjects) or
(exists cc : ControlComponent |
  cc in controlcomponents setunion
  r_getallmethods(globalobjects) setunion
  r_getallmethods(r_getallobjects({o : Object |
    exists c : controlcomponents; n : Name |
    n in c.localobjects @ n = o.name})) @
  {c.o1.name, c.o2.name} subseteq cc.localobjects) or
(exists cc1, cc2 : controlcomponents; t1, t2 : threads |
  cc1 /= cc2 and t1 /= t2 and cc1 in t1 and cc2 in t2 @
  c.o1.name in cc1.localobjects and
c.o2.name in cc2.localobjects)) and
% method-method calls/spawns allowed only to global objects OR between local objects in the same control component OR between objects in different threads
((c.c1.method = FALSE and c.c2.method = TRUE) =>
(c.o2 in globalobjects) or
(c.o2.name in c.c1.localobjects));
% nonmethod-method calls/spawns allowed only to global objects or from a control component to a local object
(forall t : Trigger | t in triggers and t.subtype = ctype @
t.controlcomponent in controlcomponents setunion
  r_getallmethods(globalobjects) setunion
  r_getallmethods(r_getallobjects({o : Object |
    exists c : controlcomponents; n : Name |
    n in c.localobjects @ n = o.name})) and
% trigger control component part of the system
t.outcalls subseteq controlcomponents and
% trigger out calls part of the system
t.outspawns subseteq controlcomponents and
% trigger out calls part of the system
t.inmessage in recognizedmessages and
% incoming message is a recognized message
t.outmessageset subseteq recognizedmessages and
% outgoing messages are recognized messages
(forall c2 : ControlComponent | c2 in t.outcalls @
((exists layer : Layer | layer in call_layers @
  (t.controlcomponent in first layer and c2 in second layer) or
  (c2 in first layer and t.controlcomponent in second layer))
or
(forall layer : Layer | layer in call_layers @
  {t.controlcomponent, c2} setint second layer = {}))) and
% out calls constrained to between layers or not constrained at all
(forall c2 : ControlComponent | c2 in t.outspawns @
((exists layer : Layer | layer in spawn_layers @
  (t.controlcomponent in first layer and c2 in second layer) or
  (c2 in first layer and t.controlcomponent in second layer))
or
(forall layer : Layer | layer in spawn_layers @
  {t.controlcomponent, c2} setint second layer = {}))) and
% spawn constrains to between layers or not constrained at all
(t.controlcomponent in first layer and c2 in second layer) or
(c2 in first layer and t.controlcomponent in second layer))

or
(forall layer : Layer | layer in spawn_layers @
{t.controlcomponent, c2} setint second layer = {})))));

% outspawns constrained to between layers or not constrained at all
(forall t : Trigger | t in triggers and t.subtype = otype @
t.object in r.getallobjects(globalobjects) setunion
r.getallobjects({o : Object | exists c : controlcomponents;
   n : Name | n in c.localobjects @ n = o.name}) and
% trigger object part of the system
t.outcalls subseteq controlcomponents and
% trigger outcalls part of the system
t.outspawns subseteq controlcomponents and
% trigger outspawns part of the system
t.inmessage in recognizedmessages and
% incoming message is a recognized message
t.outmessageset subseteq recognizedmessages);
% outgoing messages are recognized messages
{n : Node |
(exists c : ControlComponent | c in controlcomponents setunion
   r.getallmethods(globalobjects) setunion
   r.getallmethods(r.getallobjects({o : Object |
      exists c : controlcomponents; n : Name |
      n in c.localobjects @ n = o.name})) @ n in c.nodes) or
(exists call : ControlConnector | call in calls @ n in call.nodes) or
(exists spawn : ControlConnector | spawn in spawns @
   n in spawn.nodes) or
(exists dc : DataConnector | dc in dataconnectors @ n in dc.nodes})
subseteq nodes;
% available nodes form a superset of the connector nodes
end schema
% unique name constraints
axiom
forall p1, p2 : Port @ p1.name = p2.name => p1 = p2;
forall d1, d2 : DataComponent @ d1.name = d2.name => d1 = d2;
forall c1, c2 : ControlComponent @ c1.name = c2.name => c1 = c2;
forall d1, d2 : DataConnector @ d1.name = d2.name => d1 = d2;
forall c1, c2 : ControlConnector @ c1.name = c2.name => c1 = c2;
forall o1, o2 : Object @ o1.name = o2.name => o1 = o2;
forall t1, t2 : Trigger @ t1.name = t2.name => t1 = t2;
forall s1, s2 : System @ s1.name = s2.name => s1 = s2;
end axiom

global
getmaxthreads : System +-> P Thread;
% retrieve maximum number of different threads in a system
% Assumes that objects do not start out executing any method concurrently
axiom
forall system : System; threads : P Thread @
getmaxthreads(system) = threads <=>
(forall thread : Thread | thread in threads @
(exists topcc : ControlComponent @
   (topcc in system.initialcontrolcomponents or
    (exists t : system.triggers @ topcc in t.outspawns or
     topcc in t.outcalls) or
    (exists c1 : ControlComponent; spawn : ControlConnector |
     spawn in system.spawns @
     spawn.c1 = c1 and spawn.c2 = topcc)) and
   thread = {x : ControlComponent |
    exists chain : P ControlConnector |
    chain subseteq system.calls @
    dconnected (topcc, x) = {a,b : ControlComponent |
     exists call : ControlConnector |
     call in chain @ a = call.c1 and b = call.c2}}));
end axiom

% three weakly constrained styles

global
SoftwareBusSystems : P System;
DistributedSystems : P System;
MultithreadedSystems : P System;

axiom
forall s : System | s in SoftwareBusSystems @
(u star s.controlcomponents = {x, y : ControlComponent |
    exists d : DataConnector @ d in s.dataconnectors and
    d.c1 = x and d.c2 = y}) and
(# s.dataconnectors = # s.controlcomponents - 1);
% dataconnectors form a star (center/bus distributes messages)
forall s : System | s in DistributedSystems @
# {n : Node |
    (exists c : ControlComponent | c in s.controlcomponents @
     n in c.nodes) or
    (exists call : ControlConnector | call in s.calls @ n in call.nodes) or
    (exists spawn : ControlConnector | spawn in s.spawns @
     n in spawn.nodes) or
    (exists dc : DataConnector | dc in s.dataconnectors @
     n in dc.nodes}) > 1;
% the system rests on more than one node
forall s : System | s in MultithreadedSystems @
# (getmaxthreads(s)) > 1;
% the maximum number of possible threads is greater than one
end axiom

% Pipe & Filter Style

global
PF_Socket : P Port;
PF_Pipe : P DataConnector;
PF_Filter : P ControlComponent;
PF_System : P System;

axiom
forall s : PF_Socket @
    s.iotype /= io_inout and
    % no socket is used for both input and output
    s.streaming = TRUE;
    % all data transfers are streamed
forall p : PF_Pipe @
    p.cc_or_obj = ctype and
    % pipes always connect filters and not objects
    p.directionality = forward and
    % unidirectional
    {p.c1, p.c2} subseteq PF_Filter and
    % pipe connects two filters
    {p.p1, p.p2} subseteq PF_Socket and
    % ports belong to the P/F style
    p.p1.iotype = io_out and p.p2.iotype = io_in and
    % p1 acts as sender, p2 acts as receiver
    p.p1 /= p.p2;
    % the sender socket is different than the receiver
forall f : PF_Filter @
    f.ports subseteq PF_Socket and
    % ports belong to the P/F style
    f.localobjects = {};
forall s : PF_System @
    s.controlcomponents subseteq PF_Filter and
    s.dataconnectors subseteq PF_Pipe and
    s.globalobjects = {} and
    s.classes = {} and
    s.calls = {} and
    s.spawns = {} and
    s.shareddata = {} and
    s.call_layers = {} and
    s.spawn_layers = {} and
    s.triggers = {} and
    s.initialcontrolcomponents = s.controlcomponents and
    s.initialdataconnectors = s.dataconnectors and
    (forall p : s.dataconnectors @
        exists f1, f2 : s.controlcomponents @
            p.p1 in f1.ports and p.p2 in f2.ports) and
    % all pipe sockets connect filter sockets
    (forall p1, p2 : s.dataconnectors @
        ((p1.p1 = p2.p1 and p1.p2 = p2.p2) => p1 = p2));
    % no two pipes connect the same sender and receiver
end axiom

% Main/Subroutine Style

global
    MS_DataStructure : P DataComponent;
    MS_Procedure : P ControlComponent;
    MS_ProcedureCall : P ControlConnector;
    MS_System : P System;

axiom
forall p : MS_Procedure @
  p.ports = {} and
  p.locals subseteq MS_DataStructure and
  p.localobjects = {};
forall pc : MS_ProcedureCall @
  {pc.c1, pc.c2} subseteq MS_Procedure and
  % call connects two procedures
  pc.queuesize = 0 and
  % procedure calls cannot be queued
  pc.blocking = TRUE;
  % the calling procedure blocks until the called procedure returns
forall s : MS_System @
  s.controlcomponents subseteq MS_Procedure and
  dom s.shareddata subseteq MS_DataStructure and
  s.calls subseteq MS_ProcedureCall and
  s.globalobjects = {} and
  s.classes = {} and
  s.spawns = {} and
  s.dataconnectors = {} and
  s.spawn_layers = {} and
  s.dataconnector_layers = {} and
  s.triggers = {} and
  % s.initialcontrolcomponents = 1 and
  # (getmaxthreads(s)) = 1 and
  % all M/S systems have a single thread
  (exists main : MS_Procedure; subroutines : P MS_Procedure @
    s.initialcontrolcomponents = {main} and
    {main} setunion subroutines = s.controlcomponents and
    main notin subroutines and
    (forall call : s.calls @ call.c2 /= main) and
    cdgraph (main, subroutines) = {c1, c2 : MS_Procedure |
      exists call : s.calls @ call.c1 = c2 and call.c2 = c2}) and
  % system forms a connected, directed graph of calls starting from main
  # {n : Node | (exists c : ControlComponent |
    c in s.controlcomponents @ n in c.nodes) or
    (exists call : ControlConnector | call in s.calls @
      n in call.nodes)) = 1;
  % the system rests on one node
end axiom

% Distributed Processes Style
global
  DP_DataStructure : P DataComponent;
  DP_Socket : P Port;
  DP_Process : P ControlComponent;
  DP_Circuit : P DataConnector;
  DP_ProcessCall : P ControlConnector;
  DP_ProcessSpawn : P ControlConnector;
  DP_System : P System;
axiom
  forall s : DP_Socket @
s.iotype = io_inout;
forall p : DP_Process @
  p.ports subseteq DP_Socket and
  p.locals subseteq DP_DataStructure and
  p.localobjects = {};
forall c : DP_Circuit @
  {c.p1, c.p2} subseteq DP_Socket and
  {c.c1, c.c2} subseteq DP_Process and
  c.cc_or_obj = ctype and
  % circuits always connect processes and not objects
  c.directionality = twoway;
  % bidirectional
forall pc : DP_ProcessCall @
  {pc.c1, pc.c2} subseteq DP_Process and
  % call connects two processes
  pc.queuesize = 0 and
  % process calls cannot be queued
  pc.blocking = TRUE;
  % the calling process blocks until the called process returns
forall ps : DP_ProcessSpawn @
  {ps.c1, ps.c2} subseteq DP_Process and
  % spawn connects two processes
  ps.queuesize = 0 and
  % process spawn cannot be queued
  ps.blocking = FALSE;
  % the spawning process does not block
forall s : DP_System @
  s in DistributedSystems and
  s.controlcomponents subseteq DP_Process and
  s.dataconnectors subseteq DP_Circuit and
  s.calls subseteq DP_ProcessCall and
  s.spawns subseteq DP_ProcessSpawn and
  s.globalobjects = {} and
  s.classes = {} and
  s.shareddata = {} and
  s.triggers = {} and
  (forall c : s.dataconnectors @
    exists p1, p2 : s.controlcomponents @
      c.p1 in p1.ports and c.p2 in p2.ports) and
  % all circuit sockets connect process sockets
  (forall c1, c2 : s.dataconnectors @
    c1 /= c2 =>
      (not (c1.p1 = c2.p1 and c1.p2 = c2.p2)) and
      (not (c1.p1 = c2.p2 and c1.p2 = c2.p1)));% no two circuits connect the same pair of sockets
end axiom

% Event-Based (implicit invocation) Style

% global
EB_Event : P DataComponent;
EB_DataStructure : P DataComponent;
EB_Procedure : P ControlComponent;
EB_ProcedureCall : P ControlConnector;
EB_Object : P Object;
EB_Trigger : P Trigger;
EB_System : P System;

axiom
forall p : EB_Procedure @
p.locals subseteq EB_DataStructure and
p.ports = {};
forall pc : EB_ProcedureCall @
{pc.c1, pc.c2} subseteq EB_Procedure and
% call connects two procedures
pc.queuesize = 0 and
% procedure calls cannot be queued
pc.blocking = TRUE;
% the calling procedure blocks until the called procedure returns
forall o : EB_Object @
o.data subseteq EB_DataStructure and
o.methods subseteq EB_Procedure;
forall t : EB_Trigger @
t.object in EB_Object and
t.inmessage in EB_Event and
t.outmessageset subseteq EB_Event and
t.outcalls subseteq EB_Procedure and
t.subtype = otype and
% limit reception of events to objects
t.outspawns = {};
% no spawns allowed to be triggered
forall s : EB_System @
s.controlcomponents subseteq EB_Procedure and
s.globalobjects subseteq EB_Object and
s.calls subseteq EB_ProcedureCall and
s.recognizedmessages subseteq EB_Event and
s.triggers subseteq EB_Trigger and
s.spawns = {} and
s.shareddata = {} and
s.triggers /= {};
end axiom

% Helper function for recursively retrieving all the control components associated with
% a set of control components and objects. Retrieves methods of all local objects
% found.

global
globalcontrolcomponents :
(P ControlComponent & P Object) +-> P ControlComponent;

axiom
forall inputccs, outputccs : P ControlComponent; inputobs : P Object @
globalcontrolcomponents(inputccs, inputobs) = outputccs <=>
outputccs = inputccs setunion r_getallmethods(inputccs) setunion
r_getallmethods(r_getallobjects({o : Object |
exists c : inputccs; n : Name |)})
n in c.localobjects @ n = o.name));
end axiom

% Helper function for recursively retrieving all the objects associated with a set of control components and objects. Retrieves all local objects found.
global
getallobjects : (P ControlComponent & P Object) +-> P Object;
axiom
for all inputobs, outputobs : P Object; inputccs : P ControlComponent @
getallobjects(inputccs, inputobs) = outputobs <=>
outputobs = inputobs setunion
r_getallobjects({o : Object | exists c : inputccs; n : Name |
    n in c.localobjects @ n = o.name});
end axiom

% group operation for composition

global
group : (P System & Name & P ControlConnector & P ControlConnector &
(DataComponent +-> (P ControlComponent & P Object)) & P DataConnector & Layers & Layers & Layers)
+-> System;
% inputs include a set of (sub)systems and new bridging connectors. output is the resulting system.
axiom
for all subsystems : P System; newname : Name;
    newcalls, newspawns : P ControlConnector;
    extendedshareddata : DataComponent +->
        (P ControlComponent & P Object);
    newdataconnectors : P DataConnector;
    newcall_layers, newspawn_layers, newdataconnector_layers : Layers;
    newsystem : System @
group(subsystems, newname, newcalls, newspawns, extendedshareddata, newdataconnectors, newcall_layers, newspawn_layers, newdataconnector_layers) =
newsystem <=>
    newsystem.name = newname and
    newsystem.initialcontrolcomponents =
    {x : ControlComponent |
        exists s : System | s in subsystems
        @ x in s.initialcontrolcomponents} and
    newsystem.controlcomponents =
    {x : ControlComponent |
        exists s : System | s in subsystems
        @ x in s.controlcomponents} and
    newsystem.globalobjects =
    {x : Object |
        exists s : System | s in subsystems
        @ x in s.globalobjects} and
    (forall c : ControlConnector | c in newcalls or c in newspawns @
    (exists s1, s2 : subsystems @
    c.c1 in getallcontrolcomponents(s1.controlcomponents,
    s2.controlcomponents, r_getallobjects({o : Object | exists c : inputccs; n : Name |
    n in c.localobjects @ n = o.name}))));
s1.globalobjects) and
c.c2 in getallcontrolcomponents(s2.controlcomponents,
s2.globalobjects) and
s1 /= s2)) and
(forall d : DataConnector | d in newdataconnectors @
(exists s1, s2 : subsystems @
  d.c1 in getallcontrolcomponents(s1.controlcomponents,
s1.globalobjects) and
  d.c2 in getallcontrolcomponents(s2.controlcomponents,
s2.globalobjects) and
  s1 /= s2)) and
% all new connectors span the subsystems
{c : ControlComponent |
  (exists call : ControlConnector | call in newcalls @
    c = call.c1 or c = call.c2) or
  (exists spawn : ControlConnector | spawn in newspawns @
    c = spawn.c1 or c = spawn.c2) or
  (exists dc : DataConnector | dc in newdataconnectors @
    c = dc.c1 or c = dc.c2)) subseteq newsystem.controlcomponents and
% all components addressed in the new calls, spawns, and dataconnectors are
% members of the new system control components (this constraint is redundant)
(forall d : DataComponent; cset : P ControlComponent; oset : P Object | (d, (cset, oset)) in extendedshareddata @
(exists oldcset : P ControlComponent; oldoset : P Object; s : System @
  s in subsystems and (d, (oldcset, oldoset)) in s.shareddata and
  oldcset subseteq cset and
  cset subseteq getallcontrolcomponents(newsystem.controlcomponents,
  newsystem.globalobjects) and
  oldoset subseteq oset and
  oset subseteq getallobjects(newsystem.controlcomponents,
  newsystem.globalobjects)) and
% extended shareddata uses existing data but with larger sets of sharing com-
% ponents/objects
{x : ControlConnector | exists s : System | s in subsystems @ x in s.calls} setint newcalls = {} and
newsystem.calls =
{x : ControlConnector | exists s : System | s in subsystems @ x in s.calls} setunion newcalls and
{x : DataConnector | exists s : System | s in subsystems @ x in s.dataconnectors}
setint newdataconnectors = {} and
newsystem.dataconnectors = 
{x : DataConnector |
  exists s : System | s in subsystems
  @ x in s.dataconnectors} setunion newdataconnectors and
newsystem.shareddata = 
{x : DataComponent & (P ControlComponent & P Object) |
  exists s : System | s in subsystems
  @ x in s.shareddata} += extendedshareddata and
% extended shared data overrides subsystem's shared data
(forall layer : Layer | layer in newcall_layers or
layer in newspawn_layers or
layer in newdataconnector_layers @
  first layer setunion second layer subseq
  getallcontrolcomponents(newsystem.controlcomponents,
  newsystem.globalobjects)) and
% all layers drawn from the control components
newsystem.call_layers = 
{x : Layer |
  exists s : System | s in subsystems
  @ x in s.call_layers} setunion newcall_layers and
newsystem.spawn_layers = 
{x : Layer |
  exists s : System | s in subsystems
  @ x in s.spawn_layers} setunion newspawn_layers and
newsystem.dataconnector_layers = 
{x : Layer |
  exists s : System | s in subsystems
  @ x in s.dataconnector_layers} setunion newdataconnector_layers and
(forall call : ControlConnector | call in newcalls @
  ((exists layer : Layer | layer in newsystem.call_layers @
    (call.c1 in first layer and call.c2 in second layer) or
    (call.c2 in first layer and call.c1 in second layer))
or
  (forall layer : Layer | layer in newsystem.call_layers @
    {call.c1, call.c2} setint second layer = {}})) and
% calls constrained to between layers or not constrained at all
(forall spawn : ControlConnector | spawn in newspawns @
  ((exists layer : Layer | layer in newsystem.spawn_layers @
    (spawn.c1 in first layer and spawn.c2 in second layer) or
    (spawn.c2 in first layer and spawn.c1 in second layer))
or
  (forall layer : Layer | layer in newsystem.spawn_layers @
    {spawn.c1, spawn.c2} setint second layer = {}})) and
% spawns constrained to between layers or not constrained at all
(forall d : DataConnector | d in newdataconnectors @
  ((exists layer : Layer | layer in newsystem.dataconnector_layers @
    (d.c1 in first layer and d.c2 in second layer) or
    (d.c2 in first layer and d.c1 in second layer))
or
(forall layer : Layer | layer in newsystem.dataconnector_layers @
  {d.c1, d.c2} setint second layer = {}))) and
%  dataconnectors constrained to between layers or not constrained at all
(forall s1, s2 : System | s1 in subsystems and s2 in subsystems @
  s1.recognizedmessages = s2.recognizedmessages) and
%  all subsystems have identical sets of recognized messages
newsystem.recognizedmessages =
  {x : DataComponent |
    exists s : System | s in subsystems
@  x in s.recognizedmessages} and
newsystem.triggers =
  {x : Trigger |
    exists s : System | s in subsystems
@  x in s.triggers} and
newsystem.nodes =
  {x : Node |
    exists s : System | s in subsystems
@  x in s.nodes} and
not (exists threads : P Thread;
  l1, l2, l3 : Thread;
  c1, c2a, c2b, c3 : ControlComponent;
  d1, d2 : DataConnector |
    threads = {thread : Thread | thread in
      getmaxthreads(newsystem)} and
    l1 in threads and l2 in threads and l3 in threads and
    l1 /= l2 and l2 /= l3 and l3 /= l1 and c2a /= c2b and
    c1 in l1 and c2a in l2 and c2b in l2 and c3 in l3 and
    ((d1.c1 = c1 and d1.c2 = c2a) or (d1.c1 = c2a and d1.c2 = c1)) and
    ((d2.c1 = c3 and d2.c2 = c2b) or (d2.c1 = c2b and d2.c2 = c3)) @
    d1 in newsystem.dataconnectors and
    d2 in newsystem.dataconnectors) and
%  2 threads may not have dataconnectors going to 2 different components in a
  thread
not (exists threads : P Thread;
  l1, l2 : Thread;
  c1, c2 : ControlComponent;
  s1 : DataComponent & (P ControlComponent & P Object) |
  c1 in (first (second s1)) and c2 in (first (second s1)) and
    threads = {thread : Thread |
      thread in getmaxthreads(newsystem)} and
    l1 in threads and l2 in threads and
    c1 /= c2 and l1 /= l2 and c1 in l1 and c2 in l2 @
    s1 in newsystem.shareddata) and
%  2 threads may not share a piece of data
end axiom
Appendix C: Our Z Model

This is our complete model written in Z. Various parts of it were discussed separately throughout this dissertation, this appendix assembles it all in one place.

specification

generic [X]
% True iff a pair of nodes are connected by a set of edges in an undirected graph (where the edges are given as a set of binary tuples of nodes)
uconnected : (X & X) <-> (X <-> X)
where
forall a, b : X; edges : X <-> X @
  uconnected (a,b) = edges <=>
  ((exists c : X @
    % an edge exists from node a to some other node c, and nodes b and c are connected (recursive definition)
    ((a,c) in edges or (c,a) in edges) and
    uconnected (c, b) = edges)
  or
% trivial case: each node is connected to itself
  (exists a : X @ a=b))
end generic

generic [X]
% True iff a pair of nodes are connected by a set of edges in a directed graph (where the edges are given as a set of binary tuples of nodes)
dconnected : (X & X) <-> (X <-> X)
where
forall a, b : X; edges : X <-> X @
dconnected (a,b) = edges <=>
  ((exists c : X @
    % An edge exists from node a to some other node c, and nodes b and c are connected (recursive definition)
    (a,c) in edges and dconnected (c, b) = edges)
  or
% trivial case: each node is connected to itself
  (exists a : X @ a=b))
end generic

generic [X]
% True iff an undirected graph is connected (where the graph is represented as a set of nodes and a set of edges)
cugraph : P X <-> (X <-> X)
where
forall nodes : P X; edges : X <-> X @
cugraph nodes = edges <=>
% All nodes are connected to each other
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and (forall a, b : nodes @ uconnected (a,b) = edges))
or
% Trivial case: a graph of zero or one node
(#nodes = 0 or #nodes = 1))
end generic

generic [X]
% True iff a directed graph is connected with respect to a given `root' node (where
the graph is represented as a set of nodes and a set of edges)
cdgraph : (X & P X) <-> (X <-> X)
where
forall root : X; nodes : P X; edges : X <-> X | root in nodes @
cdgraph (root, nodes) = edges <=>
% All nodes are connected to the root
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and (forall a : nodes @ dconnected (root, a) = edges))
or
% Trivial case: a graph of zero or one node
(#nodes = 0 or #nodes = 1))
end generic

generic [X]
% True iff an undirected graph is a tree (where the graph is represented as a set of
nodes and a set of edges)
utree : P X <-> (X <-> X)
where
forall nodes : P X; edges : X <-> X @
utree nodes = edges <=>
% The number of edges is one less than the number of nodes, and all nodes
are connected to each other
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and #edges = #nodes - 1
and (forall a, b : nodes @ uconnected (a,b) = edges))
or
% Trivial case: a tree of zero or one node
(#nodes = 0 or #nodes = 1))
end generic

generic [X]
% True iff a directed graph is a tree with respect to a given `root' node (where the
graph is represented as a set of nodes and a set of edges)
dtree : (X & P X) <-> (X <-> X)
where
forall root : X; nodes : P X; edges : X <-> X | root in nodes @
dtree (root, nodes) = edges <=>
% The number of edges is one less than the number of nodes, and all nodes
are connected to the root
((#nodes > 1 and dom edges = nodes and ran edges = nodes
and #edges = #nodes - 1
and (forall a : nodes @ dconnected (root, a) = edges))
or
% Trivial case: a tree of zero or one node
(#nodes = 0 or #nodes = 1))
end generic

generic [X]
% True iff an undirected graph is a star
ustar : P X <-> (X <-> X)
where
forall nodes : P X; edges : X <-> X @
ustar nodes = edges
% The graph is a tree, and one node is found in all the edges
(utree nodes = edges) and
(exists center : X @
  #nodes > 1 =>
  (forall a,b : X | (a,b) in edges @ a = center or b = center)))
end generic

generic [X]
% True iff a directed graph is a star
dstar : (X & P X) <-> (X <-> X)
where
forall center : X; nodes : P X; edges : X <-> X | center in nodes @
dstar (center, nodes) = edges
% The graph is a tree, and one node is found in all the edges (the root)
(dtree (center, nodes) = edges) and
#nodes > 1 =>
(exists a,b : X | (a,b) in edges @ a = center))
end generic

generic [X]
% True iff an undirected graph is a ring
uring : P X <-> (X <-> X)
where
forall nodes : P X; edges : X <-> X @
uring nodes = edges
% The number of edges is equal to the number of nodes, and each node is
found in 2 edges
((#nodes > 2 and dom edges = nodes and ran edges = nodes
and #edges = #nodes and
(forall a : nodes @
  #{x, y : X | (x,y) in edges and (x=a or y=a)} = 2))
or
% Trivial case: a ring of zero, one, or two nodes
(#nodes = 0 or #nodes = 1 or #nodes = 2))
end generic
generic [X]
  % True iff an undirected graph is fully connected
  ufullyconnectedgraph : P X <-> (X <-> X)
where
  forall nodes : P X; edges : X <-> X @
  ufullyconnectedgraph nodes = edges <=>
  % The number of edges is equal to the number of nodes multiplied by the 
  same minus one, all divided by two, and an edge exists between all 
  pairs of nodes
  ((#nodes > 1 and dom edges = nodes and ran edges = nodes 
    and #edges = (#nodes * (#nodes - 1)) div 2 
    and (forall a,b : nodes @
         (a,b) in edges or (b,a) in edges))
  or
  % Trivial case: a fully connected graph of zero or one node
  (#nodes = 0 or #nodes = 1))
end generic

% Sets of platform options
[CpuModelType, DistributedCommunicationMechanismType, OperatingSystemType, GraphicalUserInterfaceType, ImplementationLanguageType]

% Partial listing of the possible platform options

% Given set of names for components, connectors, etc.
[Name]

[VendorType, VersionType]

schema PlatformOption [X]
  type : X;
  vendor : VendorType;
  version : VersionType;
end schema

% Schema for platform options

schema Platform
  cpumodel : PlatformOption[CpuModelType];
  discommmech : PlatformOption[DistributedCommunicationMechanismType];
language : PlatformOption[ImplementationLanguageType];
   os : PlatformOption[OperatingSystemType];
   gui : PlatformOption[GraphicalUserInterfaceType];
end schema

% inrel PlatformCompatible

global
% Determine if two platforms are compatible with each other
  _ PlatformCompatible _ : Platform <-> Platform;
axiom
   forall p1, p2 : Platform @ p1 PlatformCompatible p2 <=>
      p1.cpumodel = p2.cpumodel and
      p1.discommmech = p2.discommmech and
      p1.language = p2.language and
      p1.os = p2.os and
      p1.gui = p2.gui;
end axiom

% Time granularity type

TimeGranularity ::= seconds | miliseconds | microseconds

schema Time
   day : N;
   month : N;
   year : N;
   hour : N;
   minute : N;
   second : N;
   microsecond : N;
% Could have added here restrictions on valid dates. Will assume that the entities
   using time references already do it.
end schema

% Percent type

Percent == {n : N | n <= 100}

schema Resources
% CPU load (percent of total load)
   cpuload : Percent;
% Source lines of code (lines)
   sloc : N1;
% Data size (bytes)
   datasize : N1;
% Code size (bytes)
   codesize : N1;
% Main memory (bytes)
   mainmemory : N1;
% Network bandwidth (bytes per second)
   networkbandwidth : N1;
end schema
% inrel ResourceCompatible

global
% ‘zero’ for resources schema
NoResources : Resources;
% Adds a set of resources together
addResources : (P Resources) <-> Resources;
% Determine if one set of resources is compatible with another
_ec_resourcecompatible_ : Resources <-> Resources;

axiom
NoResources.sloc = 0;
NoResources.datasize = 0;
NoResources.codesize = 0;
NoResources.cpuload = 0;
NoResources.mainmemory = 0;
NoResources.networkbandwidth = 0;
forall inputset : P Resources; r1, r2, r3 : Resources @
  (inputset = {} => addResources inputset = NoResources) and
  (inputset = {r1} => addResources inputset = r1) and
  (r1, r2 subseteq inputset and r3 notin inputset) =>
    addResources(inputset) = addResources((inputset \ {r1, r2})
      setunion {r3}) and
    r3.sloc = r1.sloc + r2.sloc and
    r3.datasize = r1.datasize + r2.datasize and
    r3.codesize = r1.codesize + r2.codesize and
    r3.cpuload = r1.cpuload + r2.cpuload and
    r3.mainmemory = r1.mainmemory + r2.mainmemory and
    r3.networkbandwidth = r1.networkbandwidth + r2.networkbandwidth);
forall r_actual, r_limits : Resources @ r_actual ResourceCompatible r_limits <=>
  r_actual.sloc <= r_limits.sloc and
  r_actual.datasize <= r_limits.datasize and
  r_actual.codesize <= r_limits.codesize and
  r_actual.cpuload <= r_limits.cpuload and
  r_actual.mainmemory <= r_limits.mainmemory and
  r_actual.networkbandwidth <= r_limits.networkbandwidth;

end axiom

schema Node
  name : Name;
  platform : Platform;
  time_representation : TimeGranularity;
  wall_clock_time : Time;
  % Resources contained
  resources : Resources;
  where
    time_representation = seconds => wall_clock_time.microsecond = 0;
    time_representation = milliseconds => wall_clock_time.microsecond >= 1000;
    time_representation = microseconds => wall_clock_time.microsecond >= 0;
end schema

% Boolean type
Boolean ::= TRUE | FALSE
% Given set for primitive data types
[DataType]

% Declare a few common primitive data types (can be extended)
% Given set of primitive data types
global
  character, integer, float, array, structure, string : DataType;
end global

% For indicating the I/O type of a port
IoType ::= io_in | io_out | io_inout

% Schema for high-level (data) port
schema Port
  name : Name;
  % Handles input, output, or both
  iotype : IoType;
  % Format of the expected data
  dataformat : P DataType;
  % Maximum input and/or output rate
  rate : N1;
  % buffersize - 0 if not buffered
  buffersize : N;
  % True if the transfer streams data incrementally
  streaming : Boolean;
  % True if the transfer blocks on output (not applicable to io_in ports)
  blocking : Boolean;
  % Maximum connections allowed
  maximumconnections : N1;
end schema

% Declare two common ports that are not attached to any particular control component
or object
global
  stdin, stdout : Port;
axiom
  stdin.iotype = io_in;
  stdout.iotype = io_out;
end axiom

% Schema for high-level data component
schema DataComponent
  name : Name;
  % Type of the data determined by internal structure
  type : seq DataType;
  % Nodes it resides on
  nodes : P Node;
  % Resource usage
  resources : Resources;
end schema
% A useful type for indicating what type a control component has
CCtype ::= object | method | other

% A useful type for indicating possible expected response times
RTimetype ::= predictable | bounded | unbounded | cyclic

% Given set of classes
[Class]

% Schema for high-level control component
schema ControlComponent
  name : Name;
  % Indicates whether it is an object, a method or some other type of control component
  type : CCtype;
  class : Class;
  ports : P Port;
  % Expected arguments if called or spawned
  arguments : seq DataType;
  % Type of returned values when returning from a call or spawn
  returnvalues : seq DataType;
  % Priority level associated
  priority : N;
  % Flags whether it is a downloadable applet or not
  applet : Boolean;
  % Flags whether it is preemptive or not
  preemptive : Boolean;
  % Expected response time
  response_time : RTimetype;
  % Flags whether it is reentrant or not
  reentrant : Boolean;
  % Used only on reentrant control components. Describes a specific invocation of the given component.
  invocation : N;
  % Local variables
  locals : P DataComponent;
  % Local objects (given only by name - unique)
  localobjects : P Name;
  % Public methods that are part of the control component (given only by name - unique)
  public_methods : P Name;
  % Private methods that are part of the control component (given only by name - unique)
  private_methods : P Name;
  % Nodes it resides on
  nodes : P Node;
  % Resource usage
  resources : Resources;
  % Expected platform
  platform : Platform;
where
% Only reentrant control components have multiple invocations.
reentrant = FALSE => invocation = 0;
end schema

% Useful abbreviations for control components that are objects or methods
Object == {c : ControlComponent | c.type = object}
Method == {c : ControlComponent | c.type = method}

% Enforce differentiation between objects, methods, and other control components
axiom
forall o : Object @
  % It has a defined class
  (exists1 c : Class @ o.class = c and
   % It has no arguments or return values
   o.arguments = {} and
   o.returnvalues = {} and
  % All of its methods reside on object nodes, and are compatible with the
  object's platform
   (forall m_name : Name @
    m_name in (o.public_methods setunion o.private_methods) and
    (exists m : Method @
     m.name = m_name <=>
     m.nodes subseteq o.nodes and
     m.platform PlatformCompatible o.platform)) and
  % Combined method resources are compatible with object resources
  addResources {r : Resources | forall m_name : Name @
    m_name in (o.public_methods setunion o.private_methods) =>
    (exists m : Method @
     m.name = m_name and
     r = m.resources)} ResourceCompatible o.resources) and
  % All of its methods are so defined
  (forall m_name : Name @
   m_name in (o.public_methods setunion o.private_methods) <=>
   (exists m : Method @
    m.name = m_name)));
forall c : ControlComponent @ c.type /= object <=>
  % It has no associated class nor methods
  not (exists1 cl : Class @ c.class = cl) and
  c.public_methods = {} and
  c.private_methods = {};
% Ensure no methods exist without a parent object, and that all of a control compo-
% nent's local objects are actually existing objects
forall m : Method @
  (exists o : Object @ m.name in (o.private_methods setunion
    o.public_methods));
end axiom

% Enforce that the localobjects to a control component be actually existing objects
axiom
forall c : ControlComponent @
  forall lo : c.localobjects@
lo in \{\text{obj\_name} : \text{Name} \mid
\quad \exists o : \text{Object} @ o.\text{name} = \text{obj\_name}\};
end axiom

% Two functions for retrieving objects and methods

\textbf{global}
% Recursively gets all objects accessible via input objects’ methods
\text{r\_getallobjs} : \text{P Object} \leftrightarrow \text{P Object};
% Recursively gets all methods accessible via input objects’ methods
\text{r\_getallmethods} : \text{P Object} \leftrightarrow \text{P Method};

\textbf{axiom}
\forall \text{inobjects}, \text{outobjects} : \text{P Object} @
\quad \text{r\_getallobjs}(\text{inobjects}) = \text{outobjects} \iff
\quad \text{outobjects} = \text{inobjects} \cup \text{r\_getallobjs}(@\text{o : Object} |
\quad \exists \text{inob} : \text{inobjects}; \text{m} : \text{Method} |
\quad \text{m.}\text{name} \in (\text{inob.}\text{private}\_\text{methods} \cup \text{inob.}\text{public}\_\text{methods}) @
\quad \text{o.}\text{name} \in \text{m.}\text{localobjects});
\forall \text{inobjects} : \text{P Object}; \text{methods} : \text{P Method} @
\quad \text{r\_getallmethods}(\text{inobjects}) = \text{methods} \iff
\quad \text{methods} = \{\text{m} : \text{Method} | \exists \text{inob} : \text{inobjects} @
\quad \text{m.}\text{name} \in (\text{inob.}\text{private}\_\text{methods} \cup \text{inob.}\text{public}\_\text{methods}) @
\quad \text{inob.}\text{name} \in \text{m.}\text{localobjects});\}
\quad \text{setunion} \text{r\_getallmethods}(\text{o : Object} |
\quad \exists \text{inob} : \text{inobjects}; \text{m} : \text{Method} |
\quad \text{m.}\text{name} \in (\text{inob.}\text{private}\_\text{methods} \cup \text{inob.}\text{public}\_\text{methods}) @
\quad \text{o.}\text{name} \in \text{m.}\text{localobjects});\});

end axiom

% A useful type for data connectors
\textbf{Directionality ::= forward | reverse | twoway}

% Schema for high-level data connector
\textbf{schema DataConnector}
\quad \text{name} : \text{Name};
\quad \% Connected ports
\quad \text{p1, p2} : \text{Port};
\quad \% Connected control components
\quad \text{c1, c2} : \text{ControlComponent};
\quad \% Format of the data expected to be transferred
\quad \text{dataformat} : \text{P DataType};
\quad \% Maximum transfer rate
\quad \text{rate} : \text{N1};
\quad \% Indicate direction of communications
\quad \text{directionality} : \text{Directionality};
\quad \% buffersize - 0 if not buffered
\quad \text{buffersize} : \text{N};
\quad \% True if each transfer is reliably sent
\quad \text{reliable} : \text{Boolean};
\quad \% Nodes it resides on
nodes : P Node;
%    Resource usage
resources : Resources;
%    Expected platform
platform : Platform;

where
%    The connected ports belong to the specified components
p1 in c1.ports and p2 in c2.ports;
%    Format of data expected by connector is that of some data that can be handled
%    by the ports
# {d : DataType | d in dataformat and
d in p1.dataformat and
d in p2.dataformat} >= 1;
%    Allowable port combinations: in - out, inout - inout, in - inout
((p1.iotype = io_in and p2.iotype = io_out and directionality = reverse) or
(p1.iotype = io_in and p2.iotype = io_inout and directionality = reverse) or
(p1.iotype = io_in and p2.iotype = io_inout) or
(p2.iotype = io_in and p1.iotype = io_out and directionality = forward) or
(p2.iotype = io_in and p1.iotype = io_inout and directionality = forward));
%    The connector nodes form a superset of the component nodes
{n : Node | n in c1.nodes or n in c2.nodes} subseq nodes;

end schema

%    A useful type for indicating what type a control connector has
CCNtype ::= call | spawn | return | interrupt | swap_in | terminate | other_ccn

%    Schema for high-level control connector
schema ControlConnector
    name : Name;
    %    Indicates whether it is a call, spawn, return, interrupt, swap-in, terminate or
    %    some other type of control connector
type : CCNtype;
    %    Connected control components
c1, c2 : ControlComponent;
    %    Arguments passed in during a call or spawn (if any)
arguments : seq DataType;
    %    Values when returning from a call or spawn (if any)
returnvalues : seq DataType;
    %    Maximum transfer rate
rate : N1;
    %    queue_size - 0 if connector does not queue requests
queue_size : N;
    %    True if the transfer blocks waiting for a response
blocking : Boolean;
    %    True if each transfer is reliably sent
reliable : Boolean;
    %    Nodes it resides on
nodes : P Node;
    %    Resource usage
resources : Resources;
    %    Expected platform
platform : Platform;

where

% The connector nodes form a superset of the component nodes
\{n : Node | n in c1.nodes or n in c2.nodes\} subseteq nodes;
arguments = c2.arguments;
returnvalues = c2.returnvalues;

end schema

Call == \{c : ControlConnector |
    c.type = call and
    c.blocking = TRUE\}

Spawn == \{c : ControlConnector |
    c.type = spawn and
    c.blocking = FALSE\}

Return == \{c : ControlConnector |
    c.type = return and
    c.blocking = TRUE and
    c.arguments = {} and
    c.queuesize = 0\}

Interrupt == \{c : ControlConnector |
    c.type = interrupt and
    c.c2.preemptive = TRUE and
    c.arguments = {} and
    c.returnvalues = {} and
    c.queuesize = 0\}

Swap_In == \{c : ControlConnector |
    c.type = swap_in and
    c.c2.preemptive = TRUE and
    c.arguments = {} and
    c.returnvalues = {} and
    c.queuesize = 0\}

Terminate == \{c : ControlConnector |
    c.type = terminate and
    c.arguments = {} and
    c.returnvalues = {} and
    c.queuesize = 0\}

% Associate the Call and Return pairs
CallReturn == Call & Return

% Associate the Spawn and Return pairs
SpawnReturn == Spawn & Return

% Associate the Interrupt and Swap_In pairs
InterruptSwapIn == Interrupt & Swap_In
axiom
% Impose artificial constraint: no spawns to object methods
forall c : Spawn @ c.c2.type /= method;
end axiom

% ControlTransfer is a Z abbreviation definition for a control connector and the 2 control components it can transfer control between. The effects of a control transfer are reflected in the schema ActualControlTransfer.
ControlTransfer == ControlConnector & (ControlComponent & ControlComponent)

% DataTransfer is a Z abbreviation definition for a data connector, the 2 control components it can transfer data between, and the various data components that can be sent in the specific situation. The effects of a data transfer are reflected in the schema ActualDataTransfer.
DataTransfer == DataConnector & (ControlComponent & ControlComponent & P DataComponent)

% A useful type for indicating what type a basic event has
BEtype ::= control | data | time

% A basic event is the description of a single event.
schema BasicEvent
  name : Name;
  basic_event_type : BEtype;
  controltransfer : ControlTransfer;
  datatransfer : DataTransfer;
  absolute_time : Time;
end schema

% Actual events are sequences of basic events.
Event == seq BasicEvent

% Function that makes sure the basic events composing a given event (sequence) belong to either one of the given basic event sets or to some time checking event.
global
% Recursively checks each of the basic events
check_basicevents : (Event & P ControlTransfer & P DataTransfer) +-> Boolean;
axiom
forall e : Event; ct : P ControlTransfer; dt : P DataTransfer @
  check_basicevents(e, ct, dt) = TRUE <=>
    ((check_basicevents((tail e), ct, dt) = TRUE and
    (((head e).basic_event_type = control and
    (head e).controltransfer in ct) or
    ((head e).basic_event_type = data and
    (head e).datatransfer in dt) or
    (head e).basic_event_type = time)) or
    (#e) = 0)
end axiom

% A useful type for indicating what type a basic action has. Actions are the results of
triggers being fired by the occurrence of some event.

BAtype == BEtype setminus \{time\}

% A basic action is the description of a single action.
schema BasicAction
  name : Name;
  basic_action_type : BAtype;
  controltransfer : ControlTransfer;
  datatransfer : DataTransfer;
end schema

% Actual actions are sequences of basic actions.
Action == seq BasicAction

% Function that makes sure the basic actions composing a given action (sequence) belong to either one of the given basic action sets.
global
  % Recursively checks each of the basic actions
  check_basicactions : (Action & P ControlTransfer & P DataTransfer) +-> Boolean;
axiom
  forall a : Action; ct : P ControlTransfer; dt : P DataTransfer @
  check_basicactions(a, ct, dt) = TRUE <=>
    ((check_basicactions((tail a), ct, dt) = TRUE and
      (head a).basic_action_type = control and
      (head a).controltransfer in ct) or
    ((head a).basic_action_type = data and
      (head a).datatransfer in dt)) or
    (#a) = 0)
end axiom

% Function that makes sure the basic actions composing a given action (sequence) are generated by the given control component.
global
  % Recursively checks each of the basic actions to make sure that the given control component was the entity generating the actual action.
  generates_basicactions : (ControlComponent & Action) +-> Boolean;
axiom
  forall cc: ControlComponent; a : Action @
  generates_basicactions(cc, a) = TRUE <=>
    ((generates_basicactions(cc, (tail a)) = TRUE and
      % If it is a control transfer then we are checking for its originating control component
      ((head a).basic_action_type = control and
        (exists cn : ControlConnector; cm2 : ControlComponent @
          (cn, (cc, cm2)) = (head a).controltransfer)) or
      % If it is a data transfer then we are checking for its originating control component
      ((head a).basic_action_type = data and
        (exists dcn : DataConnector; cm2 : ControlComponent;
          dcm : P DataComponent @
          (dcn, (cc, cm2, dcm)) = (head a).datatransfer)))) or
end axiom

% Triggers are pairs of set of basic events and the corresponding actions that they trigger.
Trigger == Event & Action

% Function that returns all the triggered control components, regardless of the kind of action (actually, the parameters are a sequence of basic action and the kind of control transfer we are looking for).
global
% Recursively checks each of the basic actions
gettriggeredcontrolcmp : (CCNType & Action) +-> P ControlComponent;
axiom
forall ccn_type : CCNType; a : Action; control_comp : P ControlComponent @
% The control components related with the first basic action union the control components related with all other basic actions
gettriggeredcontrolcmp(ccn_type, a) = control_comp setunion
gettriggeredcontrolcmp(ccn_type, tail a) =>
% Iff we are dealing with an action of control transfer with the specific type we are looking for
((head a).basic_action_type = control and
control_comp = {c : ControlComponent |
(exists ccn : ControlConnector; c1 : ControlComponent |
ccn_type = ccn.type @
(ccn, (c1, c)) = (head a).controltransfer))}
or
% The control components related with the other basic actions (excluding the first one)
gettriggeredcontrolcmp(ccn_type, a) = gettriggeredcontrolcmp(ccn_type, tail a)

=>
% Iff we are dealing with an action that is not of control transfer or has a type different than the specific one we are looking for
((head a).basic_action_type /= control or
# {c : ControlComponent |
(exists ccn : ControlConnector; c1 : ControlComponent |
ccn_type = ccn.type @
(ccn, (c1, c)) = (head a).controltransfer}) = 0);
end axiom

% SharedData is a Z abbreviation definition for a data component, and the control components that share it.
SharedData == DataComponent & P ControlComponent

% Repository is the definition for a set of data components, and the control components that have access to it.
schema Repository
name : Name;
data : P DataComponent;
accessingcmps : P ControlComponent;
persistent : Boolean;
end schema

% Layer is a Z abbreviation definition for two sets of control components (top half and bottom half of layer)
Layer == P ControlComponent & P ControlComponent

% An abbreviation definition for a set of layers
Layers == P ControlComponent <-> P ControlComponent

% A thread is a set of control components that are presumably connected by calls (not spawns). "Thread" should only be used in the proper context otherwise use "P ControlComponent".
Thread == P ControlComponent

% A useful type for indicating the kinds of reconfiguration supported by systems
Reconfiguration_type ::= online | offline | on_the_fly_garbage_collection

% Given set of synchronization mechanisms
[SynchronizationMechanism]

% Schema for high-level system
schema System
  name : Name;
  % Control unit
  controlunit : P ControlComponent;
  % Initially running control components
  initialcontrolcmps : P ControlComponent;
  % Initially running data connectors
  initialdataconnectors : P DataConnector;
  % Global objects
  globalobjs : P Object;
  % All system control components, excluding object methods
  controlcomponents : P ControlComponent;
  % Sets of control components that are running, inactive, or suspended, respectively
  runningcontrolcmps : P ControlComponent;
  inactivecontrolcmps : P ControlComponent;
  suspendedcontrolcmps : P ControlComponent;
  % Allowed system classes
  classes : P Class;
  % All system shared data
  shareddata : P SharedData;
  % All system repository
  repositories : P Repository;
  % All system data connectors
  dataconnectors : P DataConnector;
  % All system calls
  calls : P Call;
  % All system spawns
  spawns : P Spawn;
  % All system returns
returns : P Return;
% All system interrupts
interrupts : P Interrupt;
% All system swap_ins
swap_ins : P Swap_In;
% All system terminates
terminates : P Terminate;
% All system controlconnectors (contains all of the calls, spawns, returns, interrupts, swap-ins and terminates).
controlconnectors : P ControlConnector;
% All of the system pairs of calls and returns
callreturns : P CallReturn;
% All of the system pairs of spawns and returns
spawnreturns : P SpawnReturn;
% All of the system pairs of interrupts and swap_ins
interruptswapins : P InterruptSwapIn;
% All possible control transfers
controltransfers : P ControlTransfer;
% All possible data transfers
datatransfers : P DataTransfer;
% System recognized messages
recognizedmsgs : P DataComponent;
% All system recognized events
events : P Event;
% All system actions (to respond to the events)
actions : P Action;
% All system triggers
triggers : P Trigger;
% Layers with respect to calls (top, bottom)
call_layers : Layers;
% Layers with respect to spawns (top, bottom)
spawn_layers : Layers;
% Layers with respect to dataconnectors (top, bottom)
dataconnector_layers : Layers;
% Hardest expected response time
response_time : RTimetype;
% Reflects whether the system implements some form of backtracking or not.
backtracking : Boolean;
% The kinds of reconfiguration supported by the system
reconfiguration : P Reconfiguration_type;
% The global attributes of the data connectors in the system
dataconnectorattributes : DataConnector;
% The global attributes of the control connectors in the system
controlconnectorattributes : ControlConnector;
% The set of synchronization mechanisms it can use
synchronization : P SynchronizationMechanism;
% Available nodes
nodes : P Node;
% Resource usage
resources : Resources;
% Expected platform
platform : Platform;

where
% There is at least one initially running control component or global object in the system
# initialcontrolcmps + # globalobjs >= 1;
% All initially running control components belong to declared set of control components
initialcontrolcmps subseteq controlcomponents;
% The control units belong to the declared set of control components
controlunit subseteq controlcomponents;
% All global/local objects belong to a system class
forall o : Object |
  o in r_getallobjs(globalobjs) or
  o in r_getallobjs({o : Object | exists c : controlcomponents; n : Name |
    n in c.localobjects @ n = o.name}) @
  o.class in classes;
% All initially running dataconnectors belong to declared set of dataconnectors
initialdataconnectors subseteq dataconnectors;
% The sets of control components that are running, inactive, and suspended are disjoint.
runningcontrolcmps setint inactivecontrolcmps = {};
runningcontrolcmps setint suspendedcontrolcmps = {};
inactivecontrolcmps setint suspendedcontrolcmps = {};
% The system's control components are either running, inactive or suspended. There is no other possibility. Plus all of the control components in the system's running, inactive or suspended sets belong to the system.
(runningcontrolcmps setunion inactivecontrolcmps setunion suspendedcontrolcmps) = controlcomponents;
% The set of the system's control connectors is the union of its calls, spawns, returns, interrupts, swap-ins and terminates.
controlconnectors = (calls setunion spawns setunion returns setunion interrupts setunion swap_ins setunion terminates);
% All control connectors connect components that belong to the system
forall cn : ControlConnector | cn in controlconnectors @
  (exists cm1, cm2 : controlcomponents @ cm1 = cn.c1 and cm2 = cn.c2);
% All control connectors have the appropriate control transfers defined in the system.
forall ccn : ControlConnector | ccn in controlconnectors @
  (exists c1, c2 : controlcomponents @
    ((ccn, (c1, c2)) in controltransfers and
     % The connector actually connects the 2 components
     ccn.c1 = c1 and
     ccn.c2 = c2));
% All control transfers within the system deal with connectors and components of the system.
forall ccn : ControlConnector; cc1, cc2 : ControlComponent |
  (ccn, (cc1, cc2)) in controltransfers @
  ccn in controlconnectors and
  cc1 in controlcomponents and
  cc2 in controlcomponents;
% All data connectors have the appropriate data transfers defined in the system.
forall dcn : DataConnector | dcn in dataconnectors @  
(exists c1, c2 : controlcomponents; dcset : P DataComponent @  
forall dcm : dcset @ dcm in recognizedmsgs and  
(dcn, (c1, c2, dcset)) in datatransfers and  
% The connector actually connects the 2 components  
dcn.c1 = c1 and  
dcn.c2 = c2);  
% All data transfers within the system deal with data connector, and control and  
data components of the system (also, the data component must be in the  
set of recognized messages).  
forall dcn : DataConnector; cc1, cc2 : ControlComponent; dcset : P DataComponent  
|  
(dcn, (cc1, cc2, dcset)) in datatransfers @  
dcn in dataconnectors and  
cc1 in controlcomponents and  
cc2 in controlcomponents and  
(forall dcm : DataComponent | dcm in dcset @ dcm in recognizedmsgs);  
% There are no recognizedmsgs other than those defined in the system's data  
transfers.  
forall msg : DataComponent | msg in recognizedmsgs @  
(exists dcn : DataConnector; cc1, cc2 : ControlComponent;  
dcset : P DataComponent |  
(dcn, (cc1, cc2, dcset)) in datatransfers @  
msg in dcset);  
% All components connected by calls, spawns, and/or dataconnectors are mem-
bers of the system control components or of methods of global/local objects  
{c : ControlComponent |  
(exists call : calls @ c = call.c1 or c = call.c2) or  
(exists spawn : spawns @ c = spawn.c1 or c = spawn.c2) or  
(exists dc : dataconnectors @ c = dc.c1 or c = dc.c2)}  
subseteq  
controlcomponents setunion r_getallmethods(globalobjs) setunion  
r_getallmethods(r_getallobjs({o : Object | exists c : controlcomponents; n :  
Name |  
n in c.localobjects @ n = o.name}))));  
% All objects connected by dataconnectors are global or local objects  
{o : Object |  
(exists dc : dataconnectors @ o = dc.c1 or o = dc.c2})  
subseteq  
r_getallobjs(globalobjs) setunion r_getallobjs({o : Object |  
exists c : controlcomponents; n : Name | n in c.localobjects @ n = o.name}));  
% Dataconnector ports are subset of controlcomponent, method, and (global/local)  
object ports  
{s : Port | (exists dc : dataconnectors @ s = dc.p1 or s = dc.p2)} subseteq  
{s : Port | (exists c : ControlComponent | c in controlcomponents setunion  
r_getallmethods(globalobjs) setunion  
r_getallmethods(r_getallobjs({o : Object | exists c : controlcomponents;  
n : Name | n in c.localobjects @ n = o.name}))) @ s in c.ports) or  
(exists o : Object | o in r_getallobjs(globalobjs) or  
o in r_getallobjs({o : Object | exists c : controlcomponents; n : Name |}  

242
n in c.localobjects @ n = o.name)) @ s in o.ports));
% Components sharing data are found in controlcomponents/methods and in global or local objects.

forall cset : P ControlComponent |
(exists d : DataComponent @ (d, cset) in shareddata)
@cset subseteq controlcomponents setunion r_getallmethods(globalobjs) setunion

r_getallobjs({o : Object | exists c : controlcomponents; n : Name |
 n in c.localobjects @ n = o.name})) setunion

forall cset : P ControlComponent |
(exists dset : P DataComponent; r : Repository @
r in repositories and
dset = r.data and
cset = r.accessingcmps)
@cset subseteq controlcomponents setunion r_getallmethods(globalobjs) setunion

r_getallobjs({o : Object | exists c : controlcomponents; n : Name |
 n in c.localobjects @ n = o.name});
% All components accessing any of the repositories are found in controlcomponents/methods, or global or local objects

forall cset : P ControlComponent |
(exists dset : P DataComponent; r : Repository @
r in repositories and
dset = r.data and
cset = r.accessingcmps)
@cset subseteq controlcomponents setunion r_getallmethods(globalobjs) setunion

r_getallobjs({o : Object | exists c : controlcomponents; n : Name |
 n in c.localobjects @ n = o.name});
% All calls have an associated return.

forall call : Call | call in calls @
(exists return : Return @
 return in returns and
 (call, return) in callreturns and
 % Their return values match
 call.returnvalues = return.returnvalues and
 % Their c1 and c2 components correspond
 call.c1 = return.c2 and
 call.c2 = return.c1);
% All spawns have an associated return.

forall spawn : Spawn | spawn in spawns @
(exists return : Return @
 return in returns and
 (spawn, return) in spawnreturns and
 % Their return values match
 spawn.returnvalues = return.returnvalues and
 % Their c1 and c2 components correspond
 spawn.c1 = return.c2 and
 spawn.c2 = return.c1);
swap_in in swap_ins and
(interrupt, swap_in) in interruptswapins and
% Their c2 component correspond, i.e., the control component that was
interrupted is actually the one that is being swaped in.
interrupt.c2 = swap_in.c2);
% All returns must be either associated with a call or a spawn.
forall return : Return | return in returns @
(exists call : Call @
call in calls and
(call, return) in callreturns) or
(exists spawn : Spawn @
spawn in spawns and
(spawn, return) in spawnreturns);
% All swap-ins must be associated with an interrupt.
forall swap_in : Swap_In | swap_in in swap_ins @
(exists interrupt : Interrupt @
interrupt in interrupts and
(interrupt, swap_in) in interruptswapins);
% All layers drawn from the controlcomponents & methods
% Layering constraints ignore objects
forall layer : Layer | layer in call_layers or layer in spawn_layers or
layer in dataconnector_layers @
first layer setunion second layer subseteq
controlcomponents setunion r_getallmethods(globalobjs) setunion
r_getallmethods(r_getallobjs({o : Object | exists c : controlcomponents; n :
Name |
  n in c.localobjects @ n = o.name}));
% Calls constrained to between layers or not constrained at all
forall call : ControlConnector | call in calls @
((exists layer : Layer | layer in call_layers @
  (call.c1 in first layer and call.c2 in second layer) or
  (call.c2 in first layer and call.c1 in second layer))
or
  (forall layer : Layer | layer in call_layers @
   {call.c1, call.c2} setint second layer = {}));
% Spawns constrained to between layers or not constrained at all
forall spawn : ControlConnector | spawn in spawns @
((exists layer : Layer | layer in spawn_layers @
  (spawn.c1 in first layer and spawn.c2 in second layer) or
  (spawn.c2 in first layer and spawn.c1 in second layer))
or
  (forall layer : Layer | layer in spawn_layers @
   {spawn.c1, spawn.c2} setint second layer = {}));
% Dataconnectors constrained to between layers or not constrained at all
forall d : DataConnector | d in dataconnectors @
((exists layer : Layer | layer in dataconnector_layers @
  (d.c1 in first layer and d.c2 in second layer) or
  (d.c2 in first layer and d.c1 in second layer))
or
  (forall layer : Layer | layer in dataconnector_layers @
   {d.c1, d.c2} setint second layer = {}));
% A system that has predictable response time needs to have at least one of its control components with predictable response time.

response_time = predictable <=>
(exists c : ControlComponent | c in controlcomponents @ c.response_time = predictable);

% A system that has bounded response time needs to have at least one of its control components with bounded response time, and none with predictable.

response_time = bounded <=>
((exists c : ControlComponent | c in controlcomponents @ c.response_time = bounded) and
not ((exists c : ControlComponent | c in controlcomponents @ c.response_time = predictable)));

% If all control components in the system have unbounded response time, it means that the system has either unbounded or cyclic response time.

(forall c : controlcomponents @ c.response_time = unbounded) <=>
(response_time = unbounded or response_time = cyclic);

% If the system is cyclic it means that some subset of its control components form a ring with respect to its control connectors.

response_time = cyclic <=>
(exists c_set : P ControlComponent; cn_set : P ControlConnector | c_set subset controlcomponents and cn_set subset controlconnectors @
uring c_set = {a,b : ControlComponent |
exists cn : ControlConnector |
cn in cn_set @ cn.c1 = a and
cn.c2 = b});

% A system must be reconfigurable either online or offline. It may also have some on-the-fly garbage collection.

#reconfiguration = 1 or #reconfiguration = 2;
#reconfiguration = 1 =>
(reconfiguration = {online} or reconfiguration = {offline});
#reconfiguration = 2 =>
(on_the_fly_garbage_collection in reconfiguration and
(online in reconfiguration or offline in reconfiguration));
forall c : ControlComponent; threads : P Thread |
% Every control component and every method is either initial, [trigger] called, or [trigger] spawned

c in controlcomponents setunion r_getallmethods(globalobjs) setunion
r_getallmethods(r_getallobjs({o : Object | exists c : controlcomponents; n : Name |
n in c.localobjects @ n = o.name})) and
threads = {thread : Thread |
(exists topcc : ControlComponent @
topcc in initialcontrolcmps or
(exists t : triggers; e : events; a : actions | (e, a) = t @
topcc in gettriggeredcontrolcmp(call, a) or
topcc in gettriggeredcontrolcmp(spawn, a)) or
(exists c1 : ControlComponent; spawn : ControlConnector |
spawn in spawns @ spawn.c1 = c1 and spawn.c2 = topcc)) and

thread = \{x : ControlComponent |

  exists chain : P ControlConnector | chain subseteq calls @
  disconnected (topcc, x) = \{a,b : ControlComponent |
  exists call : ControlConnector |
  call in chain @ a = call.c1 and b = call.c2)\}\} @

(exists thread : Thread | thread in threads @ c in thread) and
(forall c : ControlConnector | c in calls or c in spawns @
  \%
  Method-method calls/spawns allowed only to global objects OR
  between local objects in the same controlcomponent OR between
  objects in different threads

  ((c.c1.type = method and c.c2.type = method) =>
  (c.c2 in globalobjs) or
  (exists cc : ControlComponent | cc in controlcomponents setunion
  r_getallmethods(globalobjs) setunion
  r_getallmethods(r_getallobjs({o : Object |
  exists c : controlcomponents; n : Name |
  n in c.localobjects @ n = o.name})) @
  \{c.c1.name, c.c2.name\} subseteq cc.localobjects) or
  (exists cc1, cc2 : controlcomponents; t1, t2 : threads |
  cc1 /= cc2 and t1 /= t2 and cc1 in t1 and cc2 in t2 @
  c.c1.name in cc1.localobjects and c.c2.name in cc2.localobjects))

  and

  \%
  Nonmethod-method calls/spawns allowed only to global objects or from
  a controlcomponent to a local object

  ((c.c1.type /= method and c.c2.type /= method) =>
  (c.c2 in globalobjs) or
  (c.c2.name in c.c1.localobjects)));

  \%
  The various events in the system must be composed of sequences of basic
  events that also belong to the system.

  (forall e : Event | e in events @
  check_basicevents(e, controltransfers, datatransfers) = TRUE);

  \%
  The various actions in the system must be composed of sequences of basic
  actions that also belong to the system.

  (forall a : Action | a in actions @
  check_basicactions(a, controltransfers, datatransfers) = TRUE);

  \%
  All triggers in the system must be composed of events and actions also related
  to the system.

  (forall t : Trigger | t in triggers @
  (exists e : Event; a : Action | e in events and a in actions @
  (e, a) = t));

  \%
  Available nodes form a superset of the connector nodes

  \{n : Node |

  (exists c : ControlComponent | c in controlcomponents setunion
  r_getallmethods(globalobjs) setunion
  r_getallmethods(r_getallobjs({o : Object |
  exists c : controlcomponents; n : Name |
  n in c.localobjects @ n = o.name})) @ n in c.nodes) or
  (exists call : ControlConnector | call in calls @ n in call.nodes) or
  (exists spawn : ControlConnector | spawn in spawns @ n in spawn.nodes) or
  (exists interrupt : ControlConnector | interrupt in interrupts @
  n in interrupt.nodes) or

  \%
  ...
(exists terminate : ControlConnector | terminate in terminates @ n in terminate.nodes) or
(exists dc : DataConnector | dc in dataconnectors @ n in dc.nodes)}
\subseteq \text{nodes};

% The resources on each of the system's nodes are enough to support all parts
% that reside on them.

(forall n : Node | n in nodes @
  addResources ((r : Resources |
  \% Resources required by all its data components.
  (exists d : DataComponent; c : ControlComponent; cset : \mathcal{P} \text{ControlComponent};
  dset : \mathcal{P} \text{DataComponent}; rep : Repository |
  \% Resources required by all its control components.
  (n in d.nodes and
   c in controlcomponents and
   cset \subseteq \text{controlcomponents and}
   dset \subseteq \text{controlcomponents}) @
   (d in c.locals or
    (d, cset) in \text{shareddata} or
    (d in dset and
     rep in \text{repositories} and
     dset = rep.data and
     cset = rep.accessingcmps)) and
   r = d.resources)) \cup
\% Resources required by all its control components.
\{r : Resources | (exists c : ControlComponent |
  n in c.nodes and c in controlcomponents @
  r = c.rresources)) \text{ResourceCompatible n.resources});
end schema

% Unique name constraints
axiom
  forall n1, n2 : Node @ n1.name = n2.name => n1 = n2;
  forall p1, p2 : Port @ p1.name = p2.name => p1 = p2;
  forall d1, d2 : DataComponent @ d1.name = d2.name => d1 = d2;
  \% Different invocations of the same reentrant control component do have the same
  name.
  forall c1, c2 : ControlComponent @ c1.name = c2.name =>
    c1 = c2 or
    (c1.invocation /= c2.invocation and c1.reentrant = \text{TRUE});
  forall d1, d2 : DataConnector @ d1.name = d2.name => d1 = d2;
  forall c1, c2 : ControlConnector @ c1.name = c2.name => c1 = c2;
  forall r1, r2 : Repository @ r1.name = r2.name => r1 = r2;
  forall s1, s2 : System @ s1.name = s2.name => s1 = s2;
end axiom

global
% Retrieve maximum number of different threads in a system. Assumes that
% objects do not start out executing any method concurrently
getmaxthreads : System \rightarrow \mathcal{P} \text{Thread};
axiom
  forall system : System; threads : \mathcal{P} \text{Thread} @
  getmaxthreads(system) = threads <=
(forall thread : Thread | thread in threads @
    (exists topcc : ControlComponent @
        (topcc in system.initialcontrolcmps or
         (exists t : system.triggers; e : system.events;
             a : system.actions | (e, a) = t @
             topcc in gettriggeredcontrolcmp(call, a) or
             topcc in gettriggeredcontrolcmp(spawn, a)) or
         (exists c1 : ControlComponent; spawn : ControlConnector |
             spawn in system.spawns @
             spawn.c1 = c1 and spawn.c2 = topcc)) and
        thread = {x : ControlComponent |
            exists chain : P ControlConnector | chain subseteq system.calls @
            dconnected (topcc, x) = {a,b : ControlComponent |
                exists call : ControlConnector |
                call in chain @ a = call.c1 and b = call.c2}}));
end axiom

% Three weakly constrained styles

global
SoftwareBusSystems : P System;
DistributedSystems : P System;
MultiThreadedSystems : P System;

axiom
forall s : System | s in SoftwareBusSystems @
    % Dataconnectors form a star (center/bus distributes messages)
    (# s.dataconnectors = # s.controlcomponents - 1);
forall s : System | s in DistributedSystems @
    % The system rests on more than one node
    # (n : Node |
        (exists c : ControlComponent | c in s.controlcomponents @ n in c.nodes) or
        (exists call : ControlConnector | call in s.calls @ n in call.nodes) or
        (exists spawn : ControlConnector | spawn in s.spawns @ n in spawn.nodes) or
        (exists dc : DataConnector | dc in s.dataconnectors @ n in dc.nodes)) > 1;
forall s : System | s in MultiThreadedSystems @
    % The maximum number of possible threads is greater than one
    # (getmaxthreads(s)) > 1;
end axiom

% Pipe & Filter Style

global
PF_DataStructure : P DataComponent;
PF_Socket : P Port;
PF_Pipe : P DataConnector;
PF_Filter : P ControlComponent;
PF_System : P System;

axiom
forall s : PF_Socket @
    % No socket is used for both input and output
s.iotype /= io_inout and
% All data transfers are streamed
s.streaming = TRUE;
forall p : PF_Pipe @
% Pipes always connect filters and not objects
p.c1.type = other and
p.c2.type = other and
% Unidirectional
p.directionality = forward and
% Pipe connects two filters
{p.c1, p.c2} subseteq PF_Filter and
% Ports belong to the P/F style
{p.p1, p.p2} subseteq PF_Socket and
% p1 acts as sender, p2 acts as receiver
p.p1.iotype = io_out and p.p2.iotype = io_in and
% The sender socket is different than the receiver
p.p1 /= p.p2;
forall f : PF_Filter @
% Ports belong to the P/F style
f.ports subseteq PF_Socket and
f.locals subseteq PF_DataStructure and
% There are no applets among the filters.
f.applet = FALSE and
% No filter is preemptive.
f.preemptive = FALSE and
% Filters have no associated priorities.
f.priority = 0 and
% Filters are non-reentrant.
f.reentrant = FALSE;
forall s : PF_System @
s in MultiThreadedSystems and
s.controlcomponents subseteq PF_Filter and
s.dataconnectors subseteq PF_Pipe and
s.globalobjs = {} and
s.controlconnectors = {} and
s.shareddata = {} and
s.repositories = {} and
s.triggers = {} and
s.initialcontrolcmps = s.controlcomponents and
s.initialdataconnectors = s.dataconnectors and
s.response_time = unbounded and
% PF systems have no backtracking.
s.backtracking = FALSE and
% PF systems have no control units
s.controlunit = {} and
% All pipe sockets connect filter sockets
(forall p : s.dataconnectors @
  exists f1, f2 : s.controlcomponents @
    p.p1 in f1.ports and p.p2 in f2.ports) and
% No two pipes connect the same sender and receiver
(forall p1, p2 : s.dataconnectors @
(p1.p1 = p2.p1 and p1.p2 = p2.p2) => p1 = p2));
end axiom

% Main / Subroutine Style

global
  MS_DataStructure : P DataComponent;
  MS_Procedure : P ControlComponent;
  MS_ProcedureCall : P Call;
  MS_System : P System;
axiom
  forall p : MS_Procedure @
    p.ports = {} and
    p.locals subseteq MS_DataStructure and
    % There are no applets among the procedures.
    p.applet = FALSE and
    % No procedure is preemptive.
    p.preemptive = FALSE and
    % Procedures have no associated priorities.
    p.priority = 0;
  forall pc : MS_ProcedureCall @
    % Call connects two procedures
    {pc.c1, pc.c2} subseteq MS_Procedure and
    % Procedure calls cannot be queued
    pc.queuesize = 0;
  forall s : MS_System @
    s.controlcomponents subseteq MS_Procedure and
    dom s.shareddata subseteq MS_DataStructure and
    s.repositories = {} and
    s.calls subseteq MS_ProcedureCall and
    % MS systems have only calls and returns as control connectors.
    (forall cn : ControlConnector | cn in s.controlconnectors @
     (cn.type = call or cn.type = return)) and
    s.dataconnectors = {} and
    s.triggers = {} and
    s.response_time = unbounded and
    % MS systems have no backtracking.
    s.backtracking = FALSE and
    % MS systems have no control units
    s.controlunit = {} and
    % MS systems have a single initial control component, main.
    # s.initialcontrolcmps = 1 and
    % All M/S systems have a single thread
    # (getmaxthreads(s)) = 1 and
    % System forms a connected, directed graph of calls starting from main
    (exists main : MS_Procedure; subroutines : P MS_Procedure @
     s.initialcontrolcmps = {main} and
     {main} setunion subroutines = s.controlcomponents and
     main notin subroutines and
     % Main is non-reentrant
     main.reentrant = FALSE and
     (forall call : s.calls @ call.c2 /= main) and
    ...
cdgraph (main, subroutines) = \{c1, c2 : MS_Procedure |
exists call : s.calls @ call.c1 = c2 and call.c2 = c2\}) and
% The system rests on one node
# \{n : Node |
(exists c : ControlComponent | c in s.controlcomponents @ n in c.nodes) or
(exists call : ControlConnector | call in s.calls @ n in call.nodes)\} = 1;
end axiom

% Distributed Processes Style
global
  DP_DataStructure : P DataComponent;
  DP_Socket : P Port;
  DP_Process : P ControlComponent;
  DP_Circuit : P DataConnector;
  DP_ProcessCall : P Call;
  DP_ProcessSpawn : P Spawn;
  DP_System : P System;
axiom
  forall s : DP_Socket @
    s.iotype = io_inout;
  forall p : DP_Process @
    p.ports subseteq DP_Socket and
    p.locals subseteq DP_DataStructure and
% There are no applets among the processes
    p.applet = FALSE and
% Processes have no associated priorities.
    p.priority = 0;
  forall c : DP_Circuit @
    \{c.p1, c.p2\} subseteq DP_Socket and
    \{c.c1, c.c2\} subseteq DP_Process and
% Bidirectional
    c.directionality = twoway;
  forall pc : DP_ProcessCall @
% Call connects two processes
    \{pc.c1, pc.c2\} subseteq DP_Process and
% Process calls cannot be queued
    pc.queuesize = 0;
  forall ps : DP_ProcessSpawn @
% Spawn connects two processes
    \{ps.c1, ps.c2\} subseteq DP_Process and
% Process spawn cannot be queued
    ps.queuesize = 0;
  forall s : DP_System @
    s in MultiThreadedSystems and
% The system rests on more than one node
    s in DistributedSystems and
    s.controlcomponents subseteq DP_Process and
    s.dataconnectors subseteq DP_Circuit and
% DP systems have calls, spawns, returns, and possibly interrupts (& swap-ins), and terminates.
    s.calls subseteq DP_ProcessCall and
s.spawns subseteq DP_ProcessSpawn and 
s.shareddata = {} and 
s.repositories = {} and 
s.response_time = unbounded and 
% DP systems have no backtracking. 
s.backtracking = FALSE and 
% DP systems have no control units 
s.controlunit = {} and 
% All circuit sockets connect process sockets 
(forall c : s.dataconnectors @
   exists p1, p2 : s.controlcomponents @
   c.p1 in p1.ports and c.p2 in p2.ports) and 
% No two circuits connect the same pair of sockets 
(forall c1, c2 : s.dataconnectors @
   c1 /= c2 =>
   (not (c1.p1 = c2.p1 and c1.p2 = c2.p2)) and 
   (not (c1.p1 = c2.p2 and c1.p2 = c2.p1)));

end axiom

% Event-Based (implicit invocation) Style

global
  EB_DataStructure : P DataComponent;
  EB_Procedure : P ControlComponent;
  EB_ProcedureCall : P Call;
  EB_ProcedureSpawn : P Spawn;
  EB_EventManager : P ControlComponent;
  EB_Event : P Event;
  EB_Action : P Action;
  EB_Trigger : P Trigger;
  EB_System : P System;

axiom
  forall p : EB_Procedure @
    p.locals subseteq EB_DataStructure and
    p.ports = {} and
    % There are no applets among the procedures
    p.applet = FALSE and
    % Processes have no associated priorities. They are buried within the event
    % manager and how it handles the triggering events.
    p.priority = 0;
  forall pc : EB_ProcedureCall @
    % Call connects two procedures, or the event manager and a procedure, but
    % no procedure can call the event manager
    {pc.c1, pc.c2} subseteq (EB_Procedure setunion EB_EventManager) and
    pc.c2 notin EB_EventManager and
    % Procedure calls cannot be queued
    pcqueuesize = 0;
  forall ps : EB_ProcedureSpawn @
    % Spawn connects two procedures, or the event manager and a procedure,
    % but no procedure can spawn the event manager
    {ps.c1, ps.c2} subseteq (EB_Procedure setunion EB_EventManager) and
    ps.c2 notin EB_EventManager and
% Procedure spawns cannot be queued
ps.queuesize = 0;
forall em : EB_EventManager @
  em.locals subseteq EB_DataStructure and
  % It is not an applet.
  em.applet = FALSE;
forall t : EB_Trigger @
  % All triggers are composed of events and actions that belong to the system
  (exists e : EB_Event; a : EB_Action @
   ((e, a) = t and
    % The event manager is actually initiating all triggered actions
    (exists em : EB_EventManager @
     generates_basicactions(em, a) = TRUE)));
forall s : EB_System @
  s in MultiThreadedSystems and
  s.controlcomponents subseteq (EB_Procedure setunion EB_EventManager)
  and
  dom s.shareddata subseteq EB_DataStructure and
  % There can be one or more event managers
  # {em: EB_EventManager | em in s.controlcomponents} >= 1 and
  % The event manager is always running.
  (forall em : EB_EventManager | em in s.controlcomponents @
   em in s.runningcontrolcmps) and
  % EB systems have calls, spawns, returns and possibly interrupts (& swap-ins), and terminates.
  s.calls subseteq EB_ProcedureCall and
  s.spawns subseteq EB_ProcedureSpawn and
  s.events subseteq EB_Event and
  s.actions subseteq EB_Action and
  s.triggers subseteq EB_Trigger and
  s.triggers /= {} and
  s.response_time = unbounded and
  % EB systems have no unbounded and
  s.backtracking = FALSE and
  % EB systems have their event managers as control units.
  s.controlunit = {em: EB_EventManager | em in s.controlcomponents};
end axiom

% Blackboard Style
global
  BB_DataStructure : P DataComponent;
  BB_Blackboard_Data : P DataComponent;
  BB_Ksource : P ControlComponent;
  BB_CtrlUnit : P ControlComponent;
  BB_Activation : P ControlConnector;
  BB_Terminate : P Terminate;
  BB_System : P System;
axiom
  forall bdata : BB_Blackboard_Data @
    bdata in BB_DataStructure;
  forall ksource : BB_Ksource @
ksource.ports = {} and
ksource.locals subseteq BB_DataStructure and
% There are no applets among the knowledge sources.
ksource.applet = FALSE and
% No knowledge source is preemptive.
ksource.preemptive = FALSE and
% Knowledge sources have associated priorities.
ksource.priority >= 0 and
% Knowledge sources are non-reentrant.
ksource.reentrant = FALSE;
forall cunit : BB_CtrlUnit @
cunit.ports = {} and
cunit.locals subseteq BB_DataStructure and
% The control unit is not an applet.
cunit.applet = FALSE and
% It is not preemptive.
cunit.preemptive = FALSE and
% It is non-reentrant.
cunit.reentrant = FALSE;
forall a : BB_Activation @
% Activations are made from the control unit to the various knowledge sources
a.c1 in BB_CtrlUnit and
a.c2 in BB_Ksource and
% Activations are either calls or spawns
(a.type = call or a.type = spawn) and
% Activations cannot be queued
a.queuesize = 0;
forall s : BB_System @
% There is exactly one blackboard repository
# s.repositories = 1 and
% There is exactly one control unit
# {cunit : BB_CtrlUnit | cunit in s.controlcomponents} = 1 and
% There is at least one knowledge source
# {ksource : BB_Ksource | ksource in s.controlcomponents} >= 1 and
% All knowledge sources have activations connecting to them
forall ksource : BB_Ksource |
  ksource in s.controlcomponents @
  (exists a : BB_Activation @ a.c2 = ksource)) and
% All control connectors that happen within the system are activations, returns,
and terminates.
(s.calls setunion s.spawns) subseteq BB_Activation and
s.terminates subseteq BB_Terminate and
forall cn : ControlConnector |
  cn in s.controlconnectors @
  cn.type = call or cn.type = spawn or cn.type = return or cn.type = terminate)
and
s.dataconnectors = {} and
s.shareddata = {} and
s.triggers /= {} and
% The control unit initiates all triggered actions.
forall t : Trigger | t in s.triggers @
(exists e : Event; a : Action @
  ((e, a) = t and
   (exists cunit : BB_CtrlUnit @
      generates_basicactions(cunit, a) = TRUE))) and
% BB systems have backtracking.
s.backtracking = TRUE and
% BB systems have a central control unit, the control unit.
s.controlunit = {cunit : BB_CtrlUnit | cunit in s.controlcomponents} and
# {cunit: BB_CtrlUnit | cunit in s.controlunit} = 1 and
% There is only 1 initial control component, the control unit.
# s.initialcontrolcmps = 1 and
(exists cunit : BB_CtrlUnit; ksource : P BB_Ksource @
  s.initialcontrolcmps = {cunit} and
  % All control components are either knowledge sources or the control unit.
  {cunit} setunion ksource = s.controlcomponents and
  cunit notin ksource and
  % Nobody can call or spawn the control unit.
  not (exists call : s.calls @ call.c2 = cunit) and
  not (exists spawn : s.spawns @ spawn.c2 = cunit) and
  % The system is a directed graph on a star configuration (dstar), where the
  center is the control unit, and the edges are system calls or spawns
  (activation calls or spawns to be more specific).
  dstar(cunit, ksource) = {c1, c2 : s.controlcomponents |
    exists cn : s.controlconnectors |
      (cn in s.calls or cn in s.spawns) @
      cn.c1 = c1 and cn.c2 = c2}) and
  % All blackboard data is visible to the control unit via the repository, all control
  components that can see that data belong to the system (i.e., they are
  either any of its knowledge sources or its control unit), and all control
  components that belong to the system can see the data.
  (forall r : s.repositories @
    (forall bbdata : P DataComponent |
      bbdata subseteq BB_Blackboard_Data @
      (exists cset : P ControlComponent; cunit : BB_CtrlUnit @
        bbdata = r.data and
        cset = r.accessingcmps and
        cunit in cset and
        cset = s.controlcomponents)));
end axiom
%
% Database Centric Style

global
  DB_DataStructure : P DataComponent;
  DB_Data : P DataComponent;
  DB_UsingComponent : P ControlComponent;
  DB_SupportComponent : P ControlComponent;
  DB_DBMS : P ControlComponent;
  DB_Request : P Call;
  DB_Check : P Spawn;
  DB_System : P System;
axiom
forall data : DB_Data @
    data in DB_DataStructure;
forall u : DB_UseComponent @
    u.locals subseteq DB_DataStructure;
forall s : DB_SupportComponent @
    s.locals subseteq DB_DataStructure;
forall d : DB_DBMS @
    d.locals subseteq DB_DataStructure and
    % The dbms is reentrant.
    d.reentrant = TRUE;
forall r : DB_Request @
    % Requests are made from the using and support control components to the
    DBMS.
    r.c1 in (DB_UseComponent setunion DB_SupportComponent) and
    r.c2 in DB_DBMS;
forall c : DB_Check @
    % Checks are made from the DBMS to the support control components.
    c.c1 in DB_DBMS and
    c.c2 in DB_SupportComponent;
forall s : DB_System @
    % There are one or more database repositories
    # s.repositories >= 1 and
    % There is exactly one dbms
    # (dbms : DB_DBMS | dbms in s.controlcomponents) = 1 and
    % There is at least one using control component
    # (u_cmp : DB_UseComponent | u_cmp in s.controlcomponents) >= 1 and
    % All using control components have requests associated to them
    (forall u_cmp : DB_UseComponent | u_cmp in s.controlcomponents @
        (exists r : DB_Request @ r.c1 = u_cmp)) and
    % All supporting control components have checks connecting to them
    (forall s_cmp : DB_SupportComponent | s_cmp in s.controlcomponents @
        (exists c : DB_Check @ c.c2 = s_cmp)) and
    % All control connectors that happen within the system are either checks,
    requests, their returns, and possibly interrupts (& swap-ins), and termi-
    nates.
    (s.calls setunion s.spawns) subseteq (DB_Check setunion DB_Request) and
    % There are no applets among the control components.
    (forall c : ControlComponent | c in s.controlcomponents @
        c.applet = FALSE) and
    s.shareddata = {} and
    s.triggers /= {} and
    % The DBMS initiates all triggered actions.
    (forall t : Trigger | t in s.triggers @
        (exists e : Event; a : Action @
            ((e, a) = t and
            (exists dbms : DB_DBMS @
                generates_basicactions(dbms, a) = TRUE)))) and
    s.response_time = bounded and
    % DB systems have backtracking for rollback, maintaining data integrity and
    consistency.
s.backtracking = TRUE and
% DB systems have a central control unit, the dbms.
s.controlunit = {dbms : DB_DBMS | dbms in s.controlcomponents} and
# {dbms : DB_DBMS | dbms in s.controlunit} = 1 and
(exists dbms : DB_DBMS; u_cmps : P DB_UsingComponent; s_cmps : P
DB_SupportComponent @
  % The DBMS is always running.
  dbms in s.runningcontrolcmps and
  % All control components are either the dbms, using components, or sup-
  port components.
  {dbms} setunion u_cmps setunion s_cmps = s.controlcomponents and
  dbms notin (u_cmps setunion s_cmps) and
  % The system is an undirected graph on a star configuration (ustar), where
  the center is the dbms, and the edges are system calls or spawns
  (requestes or checks).
  (exists cmps : P ControlComponent |
    (cmps subseteq s.controlcomponents) @
    ustar(cmps) = {c1, c2 : s.controlcomponents |
      exists cn : s.controlconnectors |
      (cn in s.calls or cn in s.spawns) @
      ((cn.c1 = c1 and cn.c2 = c2) or
      (cn.c1 = c2 and cn.c2 = c1)))}) and
  % All database data is visible to the dbms via the repository, all control com-
  ponents that can see that data belong to the system.
  (forall r : s.repositories @
    (forall dbdata : P DataComponent |
      dbdata subseteq DB_Data @
      (exists cset : P ControlComponent; dbms : DB_DBMS @
        dbdata = r.data and
        cset = r.accessingcmps and
        r.persistent = TRUE and
        dbms in cset and
        cset subseteq s.controlcomponents)))));
end axiom

% Rule-Based Style

global
  RB_DataStructure : P DataComponent;
  RB_Fact : P DataComponent;
  RB_Rule : P ControlComponent;
  RB_Interpreter : P ControlComponent;
  RB_Fire : P ControlConnector;
  RB_Terminate : P Terminate;
  RB_System : P System;

axiom
  forall f : RB_Fact @
    f in RB_DataStructure;
  forall r : RB_Rule @
    r.ports = {} and
    r.locals = {} and
    % Rules are not applets.
r.applet = FALSE and  
% Rules are non-preemptive.

r.preemptive = FALSE and  
% Rules have associated priorities.

r.priority >= 0 and  
% Rules are non-reentrant.

r.reentrant = FALSE;

forall i : RB_Interpreter @
  i.ports = {} and  
  i.locals = {} and  
% Interpreters are not applets.
  i.applet = FALSE and  
% Interpreters are non-preemptive.
  i.preemptive = FALSE and  
% Interpreters are non-reentrant.
  i.reentrant = FALSE;

forall f : RB_Fire @
  f.c1 in RB_Interpreter and  
  f.c2 in RB_Rule and  
% Rules firing can be either calls or spawns.
  (f.type = call or f.type = spawn);

forall t : RB_Terminate @
% The termination of rules is made by the interpreter.
  t.c1 in RB_Interpreter and  
  t.c2 in RB_Rule;

forall s : RB_System @
% There is a single repository , the working memory.
  # s.repositories = 1 and  
  % There is exactly one interpreter
  # \{ i : RB_Interpreter | i in s.controlcomponents \} = 1 and  
  % There is at least one rule
  # \{ r : RB_Rule | r in s.controlcomponents \} >= 1 and  
  % All rules have some firing mechanism associated to them.
  (forall rule : RB_Rule | rule in s.controlcomponents @  
    (exists f : RB_Fire @ f.c1 = rule)) and  
  s.dataconnectors = {} and  
  % All control connectors that happen within the system are either rules firing,
  % returning or terminating.
  (s.calls setunion s.spawns) subseteq RB_Fire and  
  s.terminates subseteq RB_Terminate and  
  (forall cn : ControlConnector | cn in s.controlconnectors @  
    cn.type = call or cn.type = spawn or cn.type = return or cn.type = terminate)

and
  s.shareddata = {} and
  s.triggers /= {} and  
  % The interpreter initiates all triggered actions.
  (forall t : Trigger | t in s.triggers @  
    (exists e : Event; a : Action @  
      ((e, a) = t and  
        (exists interp : RB_Interpreter @
generates_basicactions(interp, a) = TRUE))) and
% RB systems have backtracking to support for backward chaining.
s.backtracking = TRUE and
% RB systems have a central control unit, the interpreter.
s.controlunit = \{ i : RB_Interpreter | i in s.controlcomponents \} and
# \{ i : RB_Interpreter | i in s.controlunit \} = 1 and
% There is only 1 initial control component, the interpreter.
# s.initialcontrolcmps = 1 and
(exists interp : RB_Interpreter; rules : P RB_Rule @
  s.initialcontrolcmps = \{ interp \} and
% All control components are either the interpreter or rules.
  \{ interp \} setunion rules = s.controlcomponents and
  interp notin rules and
% The system is a directed graph on a star configuration (dstar), where the
center is the interpreter, and the edges are system calls or spawns
(fires).
  dstar(interp, rules) = \{ c1, c2 : s.controlcomponents |
    exists cn : s.controlconnectors |
      (cn in s.calls or cn in s.spawns) @
      cn.c1 = c1 and cn.c2 = c2 \} and
% All working memory data is visible to the interpreter via the repository, all
control components that can see that data belong to the system, and all
control components that belong to the system can see the data.
  (forall r : s.repositories @
    (forall fact : P DataComponent |
      fact subseteq RB_Fact @
        (exists cset : P ControlComponent; interp : RB_Interpreter @
          fact = r.data and
          cset = r.accessingcmps and
          interp in cset and
          cset = s.controlcomponents));
end axiom

% Logic-Programming Style
global
  LP_DataStructure : P DataComponent;
  LP_UnificationValue : P DataComponent;
  LP_Rule : P ControlComponent;
  LP_Interpreter : P ControlComponent;
  LP_Invoke : P ControlConnector;
  LP_Terminate : P Terminate;
  LP_System : P System;
axiom
  forall u : LP_UnificationValue @
      u in LP_DataStructure;
  forall r : LP_Rule @
      r.ports = {} and
      r.locals = {} and
% Rules are not applets.
      r.applet = FALSE and
% Rules are non-preemptive.
r.preemptive = FALSE and
%  Rules have associated priorities.
r.priority >= 0 and
%  Rules are non-reentrant.
r.reentrant = FALSE;
forall i : LP_Interpreter @
i.ports = {} and
i.locals = {} and
%  Interpreters are not applets.
i.applet = FALSE and
%  Interpreters are non-preemptive.
i.preemptive = FALSE and
%  Interpreters are non-reentrant.
i.reentrant = FALSE;
forall i : LP_Invoke @
  %  The invocation of rules is made by the interpreter.
i.c1 in LP_Interpreter and
i.c2 in LP_Rule and
  %  Rules’ invocation can be either calls or spawns.
  (i.type = call or i.type = spawn);
forall t : LP_Terminate @
  %  The termination of rules is made by the interpreter.
t.c1 in LP_Interpreter and
t.c2 in LP_Rule;
forall s : LP_System @
  %  There is a single repository.
# s.repositories = 1 and
  %  There is exactly one interpreter
# {i : LP_Interpreter | i in s.controlcomponents} = 1 and
  %  There is at least one rule
# {r : LP_Rule | r in s.controlcomponents} >= 1 and
  %  All rules have some invocation mechanism associated to them.
(forall rule : LP_Rule | rule in s.controlcomponents @
  (exists i : LP_Invoke @ i.c1 = rule)) and
s.dataconnectors = {} and
  %  All control connectors that happen within the system are either rules’ invocations, returns, or terminations.
(s.calls setunion s.spawns) subseteq LP_Invoke and
s.terminates subseteq LP_Terminate and
(forall cn : ControlConnector | cn in s.controlconnectors @
  cn.type = call or cn.type = spawn or cn.type = return or cn.type = terminate)

and
s.shareddata = {} and
s.triggers /= {} and
%  The interpreter initiates all triggered actions.
(forall t : Trigger | t in s.triggers @
  (exists e : Event; a : Action @
    ((e, a) = t and
     (exists interp : LP_Interpreter @
      generates_basicactions(interp, a) = TRUE)))) and
%  LP systems have backtracking to support for backward chaining.
s.backtracking = TRUE and
% LP systems have a central control unit, the interpreter.
s.controlunit = {i : LP_Interpreter | i in s.controlcomponents} and
# {i : LP_Interpreter | i in s.controlunit} = 1 and
% There is only 1 initial control component, the interpreter.
# s.initialcontrolcmps = 1 and
(exists interp : LP_Interpreter; rules : P LP_Rule @
  s.initialcontrolcmps = {interp} and
  % All control components are either the interpreter or rules.
  {interp} setunion rules = s.controlcomponents and
  interp notin rules and
  % The system is a directed graph on a star configuration (dstar), where the
  center is the interpreter, and the edges are system calls or spawns
  (invocations).
  dstar(interp, rules) = {c1, c2 : s.controlcomponents |
    exists cn : s.controlconnectors |
    (cn in s.calls or cn in s.spawns) @
    cn.c1 = c1 and cn.c2 = c2}) and
% All unification values are visible to the interpreter via the repository, all con-
  trol components that can see these values belong to the system, and all
  control components that belong to the system can see the values.
(forall r : s.repositories @
  (forall unifval : P DataComponent |
    unifval subseteq LP_UnificationValue @
    (exists cset : P ControlComponent; interp : LP_Interpreter @
      unifval = r.data and
      cset = r.accessingcmps and
      interp in cset and
      cset = s.controlcomponents)))));

end axiom

% Real Time Style
global
  RT_DataStructure : P DataComponent;
  RT_Task : P ControlComponent;
  RT_Scheduler : P ControlComponent;
  RT_Call : P Call;
  RT_Spawn : P Spawn;
  RT_Interrupt : P Interrupt;
  RT_Terminate : P Terminate;
  RT_System : P System;
axiom
  forall t : RT_Task @
    t.locals subseteq RT_DataStructure and
    % Tasks are not applets.
    t.applet = FALSE and
    % Tasks are preemptive.
    t.preemptive = TRUE and
    % Tasks have associated priorities.
    t.priority >= 0 and
    % Tasks are non-reentrant.
t.reentrant = FALSE;
forall s : RT_Scheduler @
  s.locals subseteq RT_DataStructure and
  % Schedulers are not applets.
s.applet = FALSE and
  % Schedulers are non-preemptive.
s.preemptive = FALSE and
  % Schedulers are non-reentrant.
s.reentrant = FALSE;
forall c : RT_Call @
  % The calls to tasks can be made by other tasks and/or the scheduler.
  (c.c1 in RT_Scheduler or c.c1 in RT_Task) and
  c.c2 in RT_Task;
forall s : RT_Spawn @
  % The spawns of tasks can be made by other tasks and/or the scheduler.
  (s.c1 in RT_Scheduler or s.c1 in RT_Task) and
  s.c2 in RT_Task;
forall i : RT_Interrupt @
  % The interrupts of tasks can be made by the scheduler only.
  i.c1 in RT_Scheduler and
  i.c2 in RT_Task;
forall t : RT_Terminate @
  % The termination of tasks is made by the scheduler.
  t.c1 in RT_Scheduler and
  t.c2 in RT_Task;
forall s : RT_System @
  s in MultiThreadedSystems and
  % There is exactly one scheduler
  # {sc : RT_Scheduler | sc in s.controlcomponents} = 1 and
  % There is at least one task
  # {t : RT_Task | t in s.controlcomponents} >= 1 and
  dom s.shareddata subseteq RT_DataStructure and
  % All tasks have some invocation mechanism associated to them.
  (forall task : RT_Task | task in s.controlcomponents @
    (exists c : RT_Call @ c.c1 = task) or
    (exists sp : RT_Spawn @ sp.c1 = task)) and
  % All control connectors that happen within the system are either tasks’ invo-
  % cations, their returns, interruptions, their swap-ins or terminations.
  s.calls subseteq RT_Call and
  s.spawns subseteq RT_Spawn and
  s.interrupts subseteq RT_Interrupt and
  s.terminates subseteq RT_Terminate and
  s.triggers /= {} and
  % The scheduler initiates all triggered actions.
  (forall t : Trigger | t in s.triggers @
    (exists e : Event; a : Action @
      ((e, a) = t and
      (exists sc : RT_Scheduler @
        generates_basicactions(sc, a) = TRUE))) and
    s.response_time = predictable and
    % RT systems have no backtracking.
s.backtracking = FALSE and
%  RT systems have a central control unit, the scheduler.
s.controlunit = {sc : RT_Scheduler | sc in s.controlcomponents} and
# {sc : RT_Scheduler | sc in s.controlunit} = 1 and
(exists sched : RT_Scheduler; tasks : P RT_Task @
  %  All control components are either the scheduler or tasks.
  {sched} setunion tasks = s.controlcomponents and
  sched notin tasks);
end axiom

%  Closed Loop Feedback Control Style
global
  CL_DataStructure : P DataComponent;
  CL_ControlledVar : P DataComponent;
  CL_ManipulatedVar : P DataComponent;
  CL_Process : P ControlComponent;
  CL_Controller : P ControlComponent;
  CL_System : P System;
axiom
  forall v : DataComponent |
    (v in CL_ControlledVar or v in CL_ManipulatedVar) @
    v in CL_DataStructure;
for all p : CL_Process @
  p.locals subseteq CL_DataStructure and
  %  There are no applets among the processes.
  p.applet = FALSE and
  %  The processes are preemptive.
  p.preemptive = TRUE and
  %  Processes are non-reentrant.
  p.reentrant = FALSE;
for all c : CL_Controller @
  c.locals subseteq CL_DataStructure and
  %  There are no applets among the controllers.
  c.applet = FALSE and
  %  Controllers are non-reentrant.
  c.reentrant = FALSE;
for all s : CL_System @
  %  There are at least one controlled variable, one manipulated variable, one
  %  processing component, and one controlling component.
  (exists datacomp : P DataComponent @
    datacomp subseteq CL_DataStructure and
    # {cv : CL_ControlledVar | cv in datacomp} >= 1 and
    # {mv : CL_ManipulatedVar | mv in datacomp} >= 1) and
    # {p : CL_Process | p in s.controlcomponents} >= 1 and
    # {c : CL_Controller | c in s.controlcomponents} >= 1 and
    %  All control components are either processing or controlling components.
    (exists proc : P CL_Process; contr : P CL_Controller @
      proc setunion contr = s.controlcomponents) and
    %  Controlled variables are explicitly shared data. Other data may also be
    %  shared.
    CL_ControlledVar subseteq dom s.shareddata and
dom s.shareddata subseteq CL_DataStructure and
% There is a cycle involving control and/or data connectors
(uring (s.controlcomponents) = {c1, c2 : s.controlcomponents |
exists cn : s.controlconnectors |
(c in s.calls or c in s.spawns) @
(cn.c1 = c1 and cn.c2 = c2) or
uring (s.controlcomponents) = {c1, c2 : s.controlcomponents |
exists dcn : s.dataconnectors @
dcn.c1 = c1 and dcn.c2 = c2}) and
s.triggers /= {} and
% CLFC systems have no backtracking.
s.backtracking = FALSE and
% CLFC systems have control unit(s), their controllers.
s.controlunit = {c : CL_Controller | c in s.controlcomponents};
end axiom

% Internet Distributed Entities Style

global
ID_DataStructure : P DataComponent;
ID_ServerData : P DataComponent;
ID_ClientData : P DataComponent;
ID_Socket : P Port;
ID_WebServer : P ControlComponent;
ID_ServerProcess : P ControlComponent;
ID_Applet : P ControlComponent;
ID_CORBAIIOPdata : P DataConnector;
ID_RMIdata : P DataConnector;
ID_HTTP : P DataConnector;
ID_Circuit : P DataConnector;
ID_CORBAIIOPcontrol : P ControlConnector;
ID_RMIcontrol : P ControlConnector;
ID_RPC : P ControlConnector;
ID_System : P System;
axiom
forall sd : ID_ServerData @
 sd in ID_DataStructure;
forall cd : ID_ClientData @
 cd in ID_DataStructure;
forall s : ID_Socket @
 s.iotype = io_inout;
forall ws : ID_WebServer @
 ws.ports subseteq ID_Socket and
 ws.locals subseteq ID_ServerData and
% There are no applets among the web servers.
 ws.applet = FALSE and
% The web server is reentrant.
 ws.reentrant = TRUE;
forall sp : ID_ServerProcess @
 sp.ports subseteq ID_Socket and
 sp.locals subseteq ID_ServerData;
forall wb : ID_WebBrowser @
  wb.ports subseteq ID_Socket and
  wb.locals subseteq ID_ClientData and
  % There are no applets among the web browsers.
  wb.applet = FALSE;
forall a : ID_Applet @
  a.ports subseteq ID_Socket and
  a.locals subseteq ID_ClientData and
  % They are applets.
  a.applet = TRUE;
forall dcn : DataConnector | (dcn in ID_CORBAIIOPdata or dcn in ID_RMIdata or
  dcn in ID_HTTP or dcn in ID_Circuit) @
  {dcn.p1, dcn.p2} subseteq DP_Socket and
  dcn.c1 in (ID_WebBrowser setunion ID_Applet) and
  dcn.c2 in (ID_WebServer setunion ID_ServerProcess) and
  % Bidirectional
  dcn.directionality = twoway;
forall ccn : ControlConnector | (ccn in ID_CORBAIIOPcontrol or
  ccn in ID_RMIcontrol or ccn in ID_RPC) @
  ccn.c1 in (ID_WebBrowser setunion ID_Applet) and
  ccn.c2 in (ID_WebServer setunion ID_ServerProcess) and
  (ccn.type = call or ccn.type = spawn);
forall s : ID_System @
  s in MultiThreadedSystems and
  % The system rests on more than one node
  s in DistributedSystems and
  s.controlcomponents subseteq (ID_WebServer setunion ID_WebBrowser set-
  union
  ID_Applet setunion ID_ServerProcess) and
  s.dataconnectors subseteq (ID_CORBAIIOPdata setunion ID_RMIdata setunion
  ID_HTTP setunion ID_Circuit) and
  s.controlconnectors subseteq (ID_CORBAIIOPcontrol setunion ID_RMIcontrol
  setunion ID_RPC) and
  % All data connectors' sockets connect control components' sockets
  (forall cn : s.dataconnectors @
    exists cm1, cm2 : s.controlcomponents @
    cn.p1 in cm1.ports and cn.p2 in cm2.ports) and
  % No two data connectors connect the same pair of sockets
  (forall c1, c2 : s.dataconnectors @
    c1 /= c2 =>
    (not (c1.p1 = c2.p1 and c1.p2 = c2.p2)) and
    (not (c1.p1 = c2.p2 and c1.p2 = c2.p1))) and
  % There is exactly one web server.
  # {ws: ID_WebServer | ws in s.controlcomponents} = 1 and
  % There are at least one web browser and one applet.
  # {wb: ID_WebBrowser | wb in s.controlcomponents} >= 1 and
  # {a: ID_Applet | a in s.controlcomponents} >= 1 and
  % Data connections are done via HTTP and at most one of: CORBA IIOP, RMI
  and socket libraries.
  # {dcn: ID_HTTP | dcn in s.dataconnectors} >= 1 and
  (# {dcn: ID_CORBAIIOPdata | dcn in s.dataconnectors} >= 1) <=>
(\# \{dc: ID_RMIdata | dc in s.dataconnectors\} = 0 and \\
\# \{dc: ID_Circuit | dc in s.dataconnectors\} = 0 and \\
(\# \{dcn: ID_RMIdata | dcn in s.dataconnectors\} >= 1) <=>
(\# \{dc: ID_CORBAIIOPdata | dc in s.dataconnectors\} = 0 and \\
\# \{dc: ID_Circuit | dc in s.dataconnectors\} = 0 and \\
(\# \{ccn: ID_CORBAIIOPcontrol | ccn in s.controlconnectors\} >= 1) <=>
(\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_CORBAIIOPcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_CORBAIIOPcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and
) <=>
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_CORBAIIOPcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and \\
\# Control connections are done via at most one of: CORBA IIOP, RMI and 
socket libraries.

(\# \{ccn: ID_CORBAIIOPcontrol | ccn in s.controlconnectors\} >= 1) <=>
(\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_CORBAIIOPcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and
) <=>
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_CORBAIIOPcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and
\# \{cc: ID_RMIcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_CORBAIIOPcontrol | cc in s.controlconnectors\} = 0 and \\
\# \{cc: ID_RPC | cc in s.controlconnectors\} = 0 and

\# Control connections are done via at most one of: CORBA IIOP, RMI and 
socket libraries.

\# System's initial^components are at least the web server and browser.
\# s.initialcontrolcmps >= 2 and 
\# s.triggers /= {} and
\# The server data resides on the web server node.
\# The client data reside on the web browsers' nodes.

(\forall \forall sd : ID_ServerData | \\
sd in (dom s.shareddata union {d : DataComponent | \\
(exists ccmp : ControlComponent | ccmp in s.controlcomponents @ 
\d in ccmp.locals}) setunion \\
\{d : DataComponent | \\
(exists dset : P DataComponent; cset : P ControlComponent; r : Reposi-
\tory | \\
r in s.repositories and 
\dset = r.data and 
cset = r.accessingcmps @ 
\d in dset()) @ 
\exists ws : ID_WebServer | ws in s.controlcomponents @ 
\sd.nodes subseteq ws.nodes)) @
\% The client data reside on the web browsers' nodes.

\forall cd : ID_ClientData | \\
cd in (dom s.shareddata setunion {d : DataComponent | \\
(exists ccmp : ControlComponent | ccmp in s.controlcomponents @ 
\d in ccmp.locals}) setunion \\
\{d : DataComponent | \\
(exists dset : P DataComponent; cset : P ControlComponent; r : Reposi-
\tory | \\
r in s.repositories and 
\dset = r.data and 
cset = r.accessingcmps @ 
\d in dset()) @
\end{document}
cd.nodes subseteq \{n : Node |
(exists wb : ID_WebBrowser | wb in s.controlcomponents @
  n in wb.nodes)\} and
\% Server processes run on the web server node.
(forall sp : ID_ServerProcess | sp in s.controlcomponents @
  (exists ws : ID_WebServer | ws in s.controlcomponents @
    sp.nodes subseteq ws.nodes)) and
\% Applets run on the web browsers’ nodes.
(forall a : ID_Applet | a in s.controlcomponents @
  a.nodes subseteq \{n : Node |
    (exists wb : ID_WebBrowser | wb in s.controlcomponents @
      n in wb.nodes)\}) and
\% ID systems have call and/or spawn and/or data connector layers.
# \{layer : Layer | layer in s.call_layers or
layer in s.spawn_layers or
layer in s.dataconnector_layers\} >= 1 and
s.response_time = unbounded and
\% ID systems have on the fly garbage collection.
on_the_fly_garbage_collection in s.reconfiguration and
\% ID systems have no control units
s.controlunit = {};
end axiom

\% Helper function for recursively retrieving all the control components associated with
a set of control components and objects. Retrieves methods of all local objects found.
global
getallcontrolcmps : (P ControlComponent & P Object) +-> P ControlComponent;
axiom
forall inputccs, outputccs : P ControlComponent; inputobs : P Object @
gequal(\getallcontrolcmps(inputccs, inputobs), outputccs) <=>
outputccs = inputccs setunion \getallmethods(inputobs) setunion
\getallmethods(r_getallobjs({o : Object | exists c : inputccs; n : Name |
  n in c.localobjects @ n = o.name}));
end axiom

\% Helper function for recursively retrieving all the objects associated with a set of control
components and objects. Retrieves all local objects found.
global
global
getallobjs : (P ControlComponent & P Object) +-> P Object;
axiom
forall inputobs, outputobs : P Object; inputccs : P ControlComponent @
gequal(\getallobjs(inputccs, inputobs), outputobs) <=>
outputobs = inputobs setunion \getallobjs({o : Object | exists c : inputccs; n : Name |
  n in c.localobjects @ n = o.name}));
end axiom

\% Helper function for recursively retrieving all the triggered calls associated with a set
of actions, each being a sequence of basic actions.
global
gettriggeredcalls : (P Action) +-> P Call;
axiom
forall actions_set : P Action; calls : P Call @
    gettriggeredcalls(actions_set) = calls <=>
    calls = \{ c : Call |
        (exists action : Action | action in actions_set @
            (exists ba : BasicAction | <<ba>> subseq action @
                ba.basic_action_type = control and
            (exists cm1, cm2 : ControlComponent @
                (c, (cm1, cm2)) = ba.controltransfer)))\};
end axiom

% Helper function for recursively retrieving all the triggered spawns associated with a
% set of actions, each being a sequence of basic actions.
global
gettriggeredspawns : (P Action) +-> P Spawn;
axiom
forall actions_set : P Action; spawns : P Spawn @
    gettriggeredspawns(actions_set) = spawns <=>
    spawns = \{ s : Spawn |
        (exists action : Action | action in actions_set @
            (exists ba : BasicAction | <<ba>> subseq action @
                ba.basic_action_type = control and
            (exists cm1, cm2 : ControlComponent @
                (s, (cm1, cm2)) = ba.controltransfer)))\};
end axiom

% Known mismatches type
Mismatch ::= M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 |
    M11 | M12 | M13 | M14 | M15 | M16 | M17 | M18 | M19 | M20 |
    M21 | M22 | M23 | M24 | M25 | M26 | M27 | M28 | M29 | M30 |
    M31 | M32 | M33 | M34 | M35 | M36 | M37 | M38 | M39 | M40 |
    M41 | M42 | M43 | M44 | M45 | M46

% Operation to check for potential mismatches
global
% Inputs include a set of (sub)systems and new bridging connectors. Output is the
% set of potential mismatches to be expected.
check_for_mismatches : (P System & P Call & P Spawn &
    P SharedData & P Repository & P DataConnector & P Trigger &
    P (P ControlComponent & Node))
    +-> P Mismatch;
axiom
forall subsystems : P System;
    bridgingcalls : P Call;
    bridgingspawns : P Spawn;
    extendedshareddata : P SharedData;
    extendedrepositories : P Repository;
    bridgingdataconnectors : P DataConnector;
    bridgingtriggers : P Trigger;
    bridgingsharedmachines : P (P ControlComponent & Node);
mismatches : P Mismatch @
check_for_mismatches(subsystems, bridgingcalls, bridgingspawns, extended-shareddata,
extendedrepositories, bridgingdataconnectors, bridgingtriggers,
bridgingsharedmachines) = mismatches <=>
% Divide bridging triggers to simplify reasoning for each mismatch.
(exists bridgtriggcalls : P Call; bridgtriggspawns : P Spawn;
bridgtriggdataxfer : P DataTransfer @
bridgtriggcalls = gettriggeredcalls(ran bridgingtriggers) and
bridgtriggspawns = gettriggeredspawns(ran bridgingtriggers) and
% Check for Mismatch 1: “Two concurrent threads share data, with potential
synchronization problems.”
% Precondition: Composing parts are to run concurrently (regardless on
whether they were originally concurrent or not), and do share some
data.
% Problem: Potential synchronization problems for concurrent threads while
sharing data.
(exendedshareddata /= {} and
% Composing parts may run concurrently
((bridgingspawns setunion bridgtriggspawns) /= {} or
(exists s : System | s in subsystems @
s in MultiThreadedSystems))) <=>
M1 in mismatches and
% Check for Mismatch 2: “A layering constraint is violated.”
% Precondition: Any of the composing subsystems originally included layering
constraints either on control or data connectors, and composition is
achieved by extending exactly that kind of connector to such a sub-
system.
% Problem: Bridging connector may ignore these constraints, thus violating
them.
% There are bridging (triggered) calls, and at least one of the subsystems
involved originally had layering constraints on calls.
(((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists s : System | s in subsystems @
s.call_layers /= {} and
(exists bc : Call | bc in (bridgingcalls setunion bridgtriggcalls) @
bc.c1 in s.controlcomponents or
bc.c2 in s.controlcomponents))) or
% There are bridging (triggered) spawns, and at least one of the subsystems
involved originally had layering constraints on spawns.
((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists s : System | s in subsystems @
s.spawn_layers /= {} and
(exists bs : Spawn | bs in (bridgingspawns setunion bridgtriggspawns) @
bs.c1 in s.controlcomponents or
bs.c2 in s.controlcomponents))) or
% There are bridging data connectors, and at least one of the subsystems
involved originally had layering constraints with respect to data connec-
(bridgingdataconnectors /= {} and
(exists s : System | s in subsystems @
s.dataconnector_layers /= {} and
(exists bd : DataConnector | bd in bridgingdataconnectors @
bd.c1 in s.controlcomponents or
bd.c2 in s.controlcomponents)))} =>
M2 in mismatches and
%
% Check for Mismatch 3: “Different sets of recognized events are used by two
subsystems that permit triggers.”
% Precondition: At least 2 of the composing subsystems permit triggers but
have different sets of possible triggering events.
% Problem: A trigger may not be recognizable by some subsystem that should.
(exists s1, s2 : System |
s1 in subsystems and
s2 in subsystems @
% Both subsystems have triggers
s1.triggers /= {} and s2.triggers /= {} and
% Their set of triggering events is different
dom s1.triggers /= dom s2.triggers) =>
M3 in mismatches and
%
% Check for Mismatch 4: “A (triggered) spawn is made into or out of a sub-
system which originally forbade them.”
% Precondition: Subsystems are being composed via (triggered) spawns, and
% at least one of the composing subsystems does not support the
dynamic creation of threads.
% Problem: A (triggered) spawn is made into or out of a subsystem that origi-
nally forbade them. May cause synchronization problems, as well as
resources contention.
% There is a bridging (triggered) spawn into or out of a subsystem that forbade
them.
((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists sp : Spawn | sp in (bridgingspawns setunion bridgtriggspawns) @
(exists s : System | s in subsystems @
(sp.c1 in s.controlcomponents or
sp.c2 in s.controlcomponents) and
s.spawns = {}))) =>
M4 in mismatches and
%
% Check for Mismatch 5: “An unrecognized triggering event is used.”
% Precondition: A trigger is used as a bridging connector, but its generating
event can never occur.
% Problem: The trigger will not cause the expected behavior, it will never fire
the related actions.
% There are bridging triggers, but at least one of them has a non-existent trig-
gering event.
(bridgingtriggers /= {} and
(exists t : Trigger; e : Event; a : Action | t in bridgingtriggers @
t = (e, a) and

270
% The event cannot occur in any of the subsystems.
(forall s : System | s in subsystems @
 check_basicevents(e, s.controltransfers, s.datatransfers) = FALSE))

<=>
M5 in mismatches and
%
% Check for Mismatch 6: “A trigger refers to subsystems which originally for-
bade triggering.”
% Precondition: All of the composing subsystems forbade triggering, and some of the composition is to be achieved via some bridging triggers.
% Problem: The trigger will go by ignored, there is no underlying mechanism available to handle the triggering.
% There are bridging triggers, and none of the original parts supported trigger-
ing.
(bridgingtriggers /= {} and
(forall s : System | s in subsystems @
s.triggers = {})) <=>
M6 in mismatches and
%
% Check for Mismatch 7: “A data connector is made into a subsystem which originally forbade them.”
% Precondition: Some of the composition is to be achieved via bridging data connectors, and at least one of the composing subsystems did not origi-

nally support data connectors.
% Problem: Unclear on how data is to be handled for sending or receiving.
% There are bridging data connectors, and at least one of them refers to a sub-
system which originally forbade them.
(bridgingdataconnectors /= {} and
(exists dc : DataConnector | dc in bridgingdataconnectors @
(exists s : System | s in subsystems @
(dc.c1 in s.controlcomponents or
dc.c2 in s.controlcomponents) and
s.dataconnectors = {}))) <=>
M7 in mismatches and
%
% Check for Mismatch 8: “A shared data relationship refers to a subsystem which originally forbade them.”
% Precondition: Some of the composition is to be achieved via bridging shared data, and one of the composing subsystems had no shared data sup-
port.
% Problem: May cause synchronization problems.
% There are bridging shared data, and at least one of the parts involved did not support that.
(extendedshareddata /= {} and
(exists cc : ControlComponent | {cc} in ran extendedshareddata @
(exists s : System | s in subsystems @
 cc in s.controlcomponents and
 s.shareddata = {}))) <=>
M8 in mismatches and
%
% Check for Mismatch 9: “A (triggered) spawn is made into or out of a sub-

...
system which is not concurrent.”

% Precondition: Some of the composition is to be achieved via bridging (triggered) spawns, and one of the composing subsystems does not support the execution of concurrent threads.

% Problem: May cause synchronization problems, as well as resources contention.

% There are bridging (triggered) spawns, and at least one of the parts involved originally had no concurrent threads.

((bridgingspawns /= {} or bridgtriggspawns /= {}) and
(exists s : System | s in subsystems @
  # (getmaxthreads(s)) = 1 and
  (exists bs : Spawn | bs in (bridgingspawns setunion bridgtriggspawns) @
   bs.c1 in s.controlcomponents or
   bs.c2 in s.controlcomponents))) <=>
M9 in mismatches and

% Check for Mismatch 10: “A remote connector is extended into or out of a non-distributed subsystem (i.e. a subsystem originally confined to a single node).”

% Precondition: One of the subsystems was originally confined to a single node, and the resulting composed system is distributed.

% Problem: The subsystem(s) originally non-distributed cannot handle delays and/or errors occurred due to some distributed communication event.

% There are bridging (triggered) calls spanning more than one node, and at least one of the subsystems involved originally was confined to a single node.

(((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists s : System | s in subsystems @
  # s.nodes = 1 and
  (exists bc : Call | bc in (bridgingcalls setunion bridgtriggcalls) @
   (bc.c1 in s.controlcomponents or
   bc.c2 in s.controlcomponents) and
   bc.c1.nodes /= bc.c2.nodes))) or
% There are bridging (triggered) spawns expaning more than one node, and at least one of the subsystems involved originally was confined to a single node.

((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists s : System | s in subsystems @
  # s.nodes = 1 and
  (exists bs : Spawn | bs in (bridgingspawns setunion bridgtriggspawns) @
   (bs.c1 in s.controlcomponents or
   bs.c2 in s.controlcomponents) and
   bs.c1.nodes /= bs.c2.nodes))) or
% There are bridging data connectors expaning more than one node, and at least one of the subsystems involved originally was confined to a single node.

(bridgingdataconnectors /= {} and
(exists s : System | s in subsystems @
  # s.nodes = 1 and
  (exists bd : DataConnector | bd in bridgingdataconnectors @
   (bd.c1 in s.controlcomponents or
   bd.c2 in s.controlcomponents)))))
bd.c2 in s.controlcomponents) and
bd.c1.nodes /= bd.c2.nodes)) or
% There are bridging shared data between components that reside in different
nodes, and at least one of the subsystems involved did not support dis-

(extendedshareddata /= {} and
(exists s : System | s in subsystems @
  # s.nodes = 1 and
(exists d : DataComponent; c1, c2 : ControlComponent; cset : P Control-
Component |
  ((d, cset) in extendedshareddata and c1 in cset and c2 in cset) @
  (c1 in s.controlcomponents or
  c2 in s.controlcomponents) and
  c1.nodes /= c2.nodes)))) <=>
M10 in mismatches and
%
% Check for Mismatch 11: “A node resource is overused.”
% Precondition: Resources required are greater than the ones available at a
given node.
% Problem: Resource overusage such as memory and disk space.
% The resources on one or more of the new system’s nodes are not enough to
support all parts that reside on it.
(exists n: Node |
(exists s : System | s in subsystems @ n in s.nodes) @
not (addResources ((r : Resources |
  % Resources required by all its subsystems data components.
  (forall s : System | s in subsystems and n in s.nodes @
  (exists d : DataComponent; c : ControlComponent;
  cset : P ControlComponent; dset : P DataComponent;
  rep : Repository |
  (n in d.nodes and
  c in s.controlcomponents and
  cset subseteq s.controlcomponents) @
  (d in c.locals or
  (d, cset) in s.shareddata or
  (d in dset and
  rep in s.Repositories and
  dset = rep.data and
  cset = rep.accessingcmps)) and
  r = d.resources))) setunion

{r : Resources |
  % Resources required by all its control components.
  (forall s : System | s in subsystems and n in s.nodes @
  (exists c : ControlComponent |
  n in c.nodes and c in s.controlcomponents @
  r = c.resources))}) ResourceCompatible n.resources)) <=>
M11 in mismatches and
%
% Check for Mismatch 12: “There is a non-deterministic set of actions that
could be caused by a trigger.”
% Precondition: At least 2 of the composing subsystems have triggering, and
there is at least one event that has more than one trigger associated with it, one from each of the composing subsystems (i.e., each of the composing subsystems had its own set of actions that should occur once the event happens).

% Problem: It is not clear which set of actions should actually occur based on the composition, and if all it is not clear what the ordering should be.

(exists s1, s2 : System |
  s1 in subsystems and
  s2 in subsystems @
  s1.triggers /= {} and
  s2.triggers /= {} and
  (exists e : Event @
    e in dom s1.triggers and
    e in dom s2.triggers)) <=>

M12 in mismatches and

% % Check for Mismatch 13: “Data connectors connecting control components that are not always active may lead into deadlock.”
% % Precondition: We have a blocking bridging data connector and the receiving control component may be inactive when data is sent through the data connector, and it may never become active (again).
% % Problem: Possibility of deadlock on the control component sending the data.

(bridgingdataconnectors /= {} and
(exists bd : DataConnector; s : System |
  s in subsystems and
  bd in bridgingdataconnectors @
    (((bd.directionality = forward or
      bd.directionality = twoway) and
      bd.p1.blocking = TRUE) or
    ((bd.directionality = reverse or
      bd.directionality = twoway) and
      bd.p2.blocking = TRUE)) and
  s.initialcontrolcmps /= s.controlcomponents and
  (bd.c1 in s.controlcomponents or
  bd.c2 in s.controlcomponents)) <=>

M13 in mismatches and

% % Check for Mismatch 14: “Data connectors connecting control components that are not always active.”
% % Precondition: We have a non-blocking and non-buffered bridging data connector and the receiving control component becomes active after the data was sent and lost.
% % Problem: Possibility of loss of data.

(bridgingdataconnectors /= {} and
(exists bd : DataConnector; s : System |
  s in subsystems and
  bd in bridgingdataconnectors @
    (((bd.directionality = forward or
      bd.directionality = twoway) and
      bd.p1.blocking = FALSE) or
    ((bd.directionality = reverse or
      bd.directionality = twoway) and
      bd.p2.blocking = FALSE)) and
  s.initialcontrolcmps /= s.controlcomponents and
  (bd.c1 in s.controlcomponents or
  bd.c2 in s.controlcomponents)) <=>

M14 in mismatches and
bd.directionality = twoway) and
bd.p2.blocking = FALSE)) and
bd.buffersize = 0 and
s.initialcontrolcmps /= s.controlcomponents and
(bd.c1 in s.controlcomponents or
bd.c2 in s.controlcomponents))) \iff M14 in mismatches and
%
% Check for Mismatch 15: “Erroneous assumption of single-thread.”
% Precondition: A control component c1 in a single-threaded subsystem calls
a control component c2 in a multi-threaded dynamic subsystem. The
problem may arise by c2 spawning c3, terminating and returning control
to c1, whose subsystem assumes it is running alone. OR there is a call
from a multi-threaded to a single-threaded subsystem. OR there is a
spawn from a (single or) multi-threaded subsystem to a single-threaded
one, the spawnee assumes it is running alone.
%
% Problem: Synchronization problems on accessing shared data, and/or
resource contention.
(((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists c : Call; s1, s2 : System |
c in (bridgingcalls setunion bridgtriggcalls) and
s1 in subsystems and
s2 in subsystems @
# (getmaxthreads(s1)) = 1 and
# (getmaxthreads(s2)) > 1 and
c.c1 in s1.controlcomponents and
c.c2 in s2.controlcomponents)) or
(((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists sp : Spawn; s1, s2 : System |
sp in (bridgingspawns setunion bridgtriggspawns) and
s1 in subsystems and
s2 in subsystems @
# (getmaxthreads(s1)) > 1 and
# (getmaxthreads(s2)) = 1 and
sp.c1 in s1.controlcomponents and
sp.c2 in s2.controlcomponents))) \iff 

M15 in mismatches and
%
% Check for Mismatch 16: “(Triggered) Call to a cyclic (non-terminating) sub-
system/control component.”
% Precondition: There is a bridging (triggered) call to a cyclic (and hence non-
terminating) subsystem / control component.
% Problem: Control will never be returned to the caller.
((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists c : Call; cm : ControlComponent |
c in (bridgingcalls setunion bridgtriggcalls) @
c.c2 = cm and
(cm.response_time = cyclic or
(exists s : System | s in subsystems @
cm in s.controlcomponents and
s.response_time = cyclic)))) \iff
M16 in mismatches and
%  
% Check for Mismatch 17: “Erroneous assumption of same underlying platform.”
% Precondition: During composition we would like to have some subsystems running on a shared machine, but they actually assume different underlying platforms.
% Problem: Cannot force fit parts to a given platform.
(bridgingsharedmachines /= {} and
(exists bm : (P ControlComponent & Node) |
  bm in bridgingsharedmachines @
  (exists c : ControlComponent; cset : P ControlComponent; n : Node |
    c in cset and
    (cset, n) = bm @
    (forall nd : Node | nd in c.nodes @
      not (nd.platform PlatformCompatible n.platform)))))) =}

M17 in mismatches and
%  
% Check for Mismatch 18: “(Triggered) Call to a private method.”
% Precondition: There is a bridging (triggered) call to a private method.
% Problem: Method not accessible to the caller.
((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists c : Call; m : Method; o : Object |
  c in (bridgingcalls setunion bridgtriggcalls) @
  c.c2 = m and
  m.name in o.private_methods)) =}

M18 in mismatches and
%  
% Check for Mismatch 19: “(Triggered) Spawn to a private method.”
% Precondition: There is a bridging (triggered) spawn to a private method.
% Problem: Method not accessible to the spawner.
((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists s : Spawn; m : Method; o : Object |
  s in (bridgingspawns setunion bridgtriggspawns) @
  s.c2 = m and
  m.name in o.private_methods)) =}

M19 in mismatches and
%  
% Check for Mismatch 20: “Sharing private data.”
% Precondition: Composition is also to be achieved via some bridging shared data, but that data is private within its original subsystem.
% Problem: Data not accessible to all of the sharing entities being composed.
(extendedshareddata /= {} and
(exists o : Object; d : DataComponent; cset : P ControlComponent |
  (d, cset) in extendedshareddata @
  d in o.locals)) =}

M20 in mismatches and
%  
% Check for Mismatch 21: “More than one central control unit exists.”
% Precondition: More than one central control unit exists as a result of composition.
% Problem: All central control units assume they have absolute control on execution sequencing.
(exists s1, s2 : System |
    s1 in subsystems and
    s2 in subsystems @
    # {s1.controlunit} = 1 and
    # {s2.controlunit} = 1) <=>
M21 in mismatches and
%
% Check for Mismatch 22: “Sharing data with a reentrant component.”
% Precondition: Composition is to be achieved via some bridging shared data, and at least one of the components to share it is reentrant.
% Problem: Sharing of data may occur with the incorrect invocation of a component.
(exendedshareddata /= {} and
(exists d : DataComponent; cset : P ControlComponent; c : ControlComponent |
    (d, cset) in extendedshareddata @
    c in cset and
    c.reentrant = TRUE)) <=>
M22 in mismatches and
%
% Check for Mismatch 23: “A reentrant component is either sending or receiving a data transfer.”
% Precondition: Composition is to be achieved via some bridging data connector, and at least one of the involved control components is reentrant.
% Problem: Potential incorrect assumption of which invocation of a component is either sending or receiving a data transfer.
(bridgingdataconnectors /= {} and
(exists dc : DataConnector |
    dc in bridgingdataconnectors @
    dc.c1.reentrant = TRUE or
    dc.c2.reentrant = TRUE)) <=>
M23 in mismatches and
%
% Check for Mismatch 24: (Triggered) Call to a non-reentrant component.
% Precondition: Composition is to be achieved via some bridging (triggered) call, and the callee is not reentrant.
% Problem: Component may already be running.
((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists c : Call | c in (bridgingcalls setunion bridgtriggcalls) @
    c.c2.reentrant = TRUE)) <=>
M24 in mismatches and
%
% Check for Mismatch 25: (Triggered) Spawn to a non-reentrant component.”
% Precondition: Composition is to be achieved via some bridging (triggered) spawn, and the spawnee is not reentrant.
% Problem: Component may already be running.
((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists s : Spawn | s in (bridgingspawns setunion bridgtriggspawns) @
    s.c2.reentrant = TRUE)) <=>
M25 in mismatches and
% Check for Mismatch 26 “Composition involves one or more prioritized parts.”
% Precondition: One of the subsystems to be composed has a prioritized structure.
% Problem: Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

(exists s1, s2 : System; c : ControlComponent |
 s1 in subsystems and
 s2 in subsystems @
 s1 /= s2 and
 c in s1.controlcomponents and
 c.priority /= 0) <=>
 M26 in mismatches and

% Check for Mismatch 27: “A prioritized system sharing a machine with some other system.”
% Precondition: A prioritized system sharing a machine with some other system.
% Problem: Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

(bridgingsharedmachines /= {} and
 (exists c : ControlComponent | c.priority /= 0 @
 (exists s1, s2 : System; c2 : ControlComponent; cset : P ControlComponent;
 n : Node |
 s1 in subsystems and
 s2 in subsystems and
 (cset, n) in bridgingsharedmachines @
 c in cset and
 c2 in cset and
 s1 /= s2 and
 c in s1.controlcomponents and
 c2 in s2.controlcomponents))) <=>
 M27 in mismatches and

% Check for Mismatch 28: “(Triggered) Call or spawn from a subsystem that may later backtrack.”
% Precondition: At least one of the callers or spawners of bridging connections belongs to a subsystem that has backtracking.
% Problem: While backtracking may cause undesired side effects on the called/spawned end.

(((bridgingcalls setunion bridgtriggcalls) /= {} and
 (exists c : Call; s : System |
 s in subsystems and
 c in (bridgingcalls setunion bridgtriggcalls) @
 c.c1 in s.controlcomponents and
 s.backtracking = TRUE)) or
 ((bridgingspawns setunion bridgtriggspawns) /= {} and
 (exists sp : Spawn; s : System |
 s in subsystems and
 sp in (bridgingspawns setunion bridgtriggspawns) @
 sp.c1 in s.controlcomponents and

 M27 in mismatches and

%
s.backtracking = TRUE)) <=>
M28 in mismatches and
%
% Check for Mismatch 29: “Data being transferred from some component(s) that may later backtrack.”
% Precondition: Composition is to be achieved via some bridging data connector, and the component to send data through such a connection belongs to a subsystem that has backtracking.
% Problem: Backtracking may cause undesired side effects on the overall composed system state.
(briddingdataconnectors /= {} and
(exists dc : DataConnector; s : System |
  s in subsystems and
  dc in bridgingdataconnectors @
  ((dc.directionality = forward or dc.directionality = twoway) and
  dc.c1 in s.controlcomponents and
  s.backtracking = TRUE) or
  ((dc.directionality = reverse or dc.directionality = twoway) and
  dc.c2 in s.controlcomponents and
  s.backtracking = TRUE))) <=>
M29 in mismatches and
%
% Check for Mismatch 30: “Sharing of data with a component(s) that may later backtrack.”
% Precondition: At least one of the subsystems sharing bridging data has backtracking.
% Problem: Backtracking may cause undesired side effects on the overall composed system state.
(extendedshareddata /= {} and
(exists d : DataComponent; cset : P ControlComponent; c : ControlComponent; s : System |
  s in subsystems and
  (d, cset) in extendedshareddata @
  c in cset and
  c in s.controlcomponents and
  s.backtracking = TRUE)) <=>
M30 in mismatches and
%
% Check for Mismatch 31: “(Triggered) Call from a subsystem requiring some predictable response times to some component(s) not originally considered.”
% Precondition: Composition is to be achieved via some bridging (triggered) call, and at least one of the calling parts requires predictable response times to some event(s).
% Problem: May have side effects on original predicted response times.
((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists c : Call; s : System |
  s in subsystems and
  c in (bridgingcalls setunion bridgtriggcalls) @
  c.c1 in s.controlcomponents and
  s.response_time = predictable)) <=>
M31 in mismatches and
%  Check for Mismatch 32: "(Triggered) Spawn to or from a subsystem requiring some predictable
response times that involves some component(s) not originally considered."
%  Precondition: Composition is to be achieved via some bridging (triggered)
spawn, and at least one of the parts involved requires predictable
response times to some event(s).
%  Problem: May have side effects on original predicted response times. The
original predicted response times did not account for concurrently executing with these other components.

((bridgingspawns setunion bridgtriggspawns) /= {} and
(exists sp : Spawn; s : System |
    s in subsystems and
    sp in (bridgingspawns setunion bridgtriggspawns) @
    (sp.c1 in s.controlcomponents or
    sp.c2 in s.controlcomponents) and
    s.response_time = predictable)) <=>

M32 in mismatches and
%  Check for Mismatch 33: "Only part of the resulting system automatically reconfigures upon failure."
%  Precondition: At least one of the parts to be composed has on-line reconfiguration, and at least one has reconfiguration off-line.
%  Problem: Only part of the resulting system automatically reconfigures upon failure.

(exists s1, s2 : System |
    s1 in subsystems and
    s2 in subsystems @
    online in s1.reconfiguration and
    offline in s2.reconfiguration) <=>

M33 in mismatches and
%  Check for Mismatch 34: "Some components that were expected to synchronize have different synchronization mechanisms."
%  Precondition: Composing parts are to run concurrently while synchronizing, and have different synchronization mechanisms.
%  Problem: Synchronization between concurrent parts cannot properly take place.

(exists s1, s2 : System |
    s1 in subsystems and
    s2 in subsystems @
    s1.synchronization /= s2.synchronization) <=>

M34 in mismatches and
%  Check for Mismatch 35: "(Triggered) Call to a component that should be preemtable and isn’t."
%  Precondition: Composition is to be achieved via a bridging (triggered) call, and the caller is preemptable, while the callee isn’t.
%  Problem: Callee will not be preempted in situations when it should.

((bridgingcalls setunion bridgtriggcalls) /= {} and
(exists c : Call; c1, c2 : ControlComponent | c in (bridgingcalls setunion bridgtriggcalls) @ c.c1 = c1 and c.c2 = c2 and c1.preemptive = TRUE and c2.preemptive = FALSE)) <=>
M35 in mismatches and
%
% Check for Mismatch 36: “(Triggered) Spawn to a component that should be preemptable and isn’t.”
% Precondition: Composition is to be achieved via a bridging (triggered) spawn, and the spawner is preemptable, while the spawnee isn’t.
% Problem: Spawnee will not be preempted in situations when it should.
((bridgingspawns setunion bridgtriggspawns) /= {} and (exists s : Spawn; c1, c2 : ControlComponent | s in (bridgingspawns setunion bridgtriggspawns) @ s.c1 = c1 and s.c2 = c2 and c1.preemptive = TRUE and c2.preemptive = FALSE)) <=>
M36 in mismatches and
%
% Check for Mismatch 37: “(Triggered) Call to a component that performs on the fly garbage collection.”
% Precondition: Composition is to be achieved via a bridging (triggered) call, the callee performs on the fly garbage collection, while the caller has requirements for some predictable or bounded response times.
% Problem: Undesirable side effects on the overall predictable or bounded response times.
((bridgingcalls setunion bridgtriggcalls) /= {} and (exists c : Call; s1, s2 : System | s1 in subsystems and s2 in subsystems and c in (bridgingcalls setunion bridgtriggcalls) @ c.c1 in s1.controlcomponents and (s1.response_time = predictable or s1.response_time = bounded) and c.c2 in s2.controlcomponents and on_the_fly_garbage_collection in s2.reconfiguration)) <=>
M37 in mismatches and
%
% Check for Mismatch 38: “Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.”
% Precondition: Composition is to be achieved via a bridging data connector, and at least one of the parts affected by such connection has encapsulation.
% Problem: Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.
(bridgingdataconnectors /= {} and (exists dc : DataConnector | dc in bridgingdataconnectors @ dc.c1.type = object or dc.c2.type = object)) <=>
M38 in mismatches and
%  
%  Check for Mismatch 39: “Sharing data with the incorrect instantiation of an
object.”
%  Precondition: Composition is to be achieved via some bridging shared data,
and at least one of the parts involved has encapsulation.
%  Problem: Sharing data with the incorrect instantiation of an object.
(exists d : DataComponent; cset : P ControlComponent; c : ControlComponent |
  (d, cset) in extendedshareddata @
  c in cset and
  c.type = object)) <=>

M39 in mismatches and
%  
%  Check for Mismatch 40: “Time represented/compared using different granu-
larities.”
%  Precondition: More than one of the composing parts has response time dif-
f erent than unbounded and cyclic unbounded. Additionally, they are dis-
tributed over machines that represent and/or compare time using
different granularities.
%  Problem: Communications concerning time cannot properly occur.
(exists s1, s2 : System; n1, n2 : Node |
  s1 in subsystems and
  s2 in subsystems and
  n1 in s1.nodes and
  n2 in s2.nodes @
  (s1.response_time /= unbounded or
  s1.response_time /= cyclic) and
  n1.time_representation /= n2.time_representation) <=>

M40 in mismatches and
%  
%  Check for Mismatch 41: “Absolute time values are off.”
%  Precondition: More than one of the composing parts has response time dif-
f erent than unbounded and cyclic unbounded. Additionally, they are dis-
tributed over machines that have their wall-clock time off.
%  Problem: Communications concerning time cannot properly occur.
(exists s1, s2 : System; n1, n2 : Node |
  s1 in subsystems and
  s2 in subsystems and
  n1 in s1.nodes and
  n2 in s2.nodes @
  (s1.response_time /= unbounded or
  s1.response_time /= cyclic) and
  n1.wall_clock_time /= n2.wall_clock_time) <=>

M41 in mismatches and
%  
%  Check for Mismatch 42: “Sharing or transferring data with differing underly-
ing representations.”
%  Precondition: Sharing or transferring data with differing underlying repre-
tsations, including differences in data formats, units and coordinate sys-
tems.
% Problem: Communications concerning the specific data will not properly occur.
(briddingdataconnectors /= {} and
(exists dc : DataConnector | dc in bridgingdataconnectors @
dc.p1.dataformat /= dc.p2.dataformat)) <=>
M42 in mismatches and
%
% Check for Mismatch 43: “Resource contention.”
% Precondition: Two or more parts are co-hosted, with at least one requiring some predictable response time, and the original analysis did not take into account some of the parts that are currently co-hosted.
% Problem: Predictable response time indirectly affected because there may be some resource contention not originally considered.
(bridgingsharedmachines /= {} and
(exists bm : (P ControlComponent & Node); s1, s2 : System | s1 in subsystems and
s2 in subsystems and
s1 /= s2 and
bm in bridgingsharedmachines @
(exists c1, c2 : ControlComponent; cset : P ControlComponent; n : Node | c1 in cset and c1 in s1.controlcomponents and
c2 in cset and c2 in s2.controlcomponents and
(cset, n) = bm @
c1.response_time = predictable))) <=>
M43 in mismatches and
%
% Check for Mismatch 44: “Potential database and/or DBMS heterogeneity problems may occur.”
% Precondition: Two or more repositories are present.
% Problem: Our model does not support for checking the specifics here, but problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.
% New parts share more than one repository.
(exists s1, s2 : System |
s1 in subsystems and
s2 in subsystems and
s1 /= s2 @
s1.repositories /= {} and
s2.repositories /= {})) <=>
M44 in mismatches and
%
% Check for Mismatch 45: “Inaccessible shared data.”
% Precondition: Composition is being achieved by having an applet share data that is local to the client side.
% Problem: Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).
% There are bridging shared data, and at least one of the parts involved is an applet, and the data resides on the client.
(extendedshareddata /= {} and
(exists cc : ControlComponent; d : DataComponent; cset : P ControlComponent; s : System; ws : ID_WebServer | s in subsystems and ws in s.controlcomponents and cc in s.controlcomponents and cc.applet = TRUE and cc in cset @ d.nodes setint ws.nodes = {} and (d, cset) in extendedsharedata)) <=>

M45 in mismatches and
%
% Check for Mismatch 46: “Distributed control units are present.”
% Precondition: Two or more of the composing subsystems have distributed
control units.
% Problem: Incorrect assumption that a problem is being solved elsewhere, by
some other control unit.
(exists s1, s2 : System | s1 in subsystems and s2 in subsystems @ # {s1.controlunit} > 1 and # {s2.controlunit} > 1) <=>

M46 in mismatches);

dend axiom

% Success boolean
Success == Boolean

% Group operation for composition
global
%
% Inputs include a set of (sub)systems and new bridging connectors. Output is the
resulting system.
group : (P System & Name & P Call & P Spawn & P SharedData & P Repository &
P DataConnector & P Trigger & P (P ControlComponent & Node) &
Layers & Layers & Layers)
+-> (Success & System);

axiom
forall subsystems : P System; newname : Name;
newcalls : P Call;
newspawns : P Spawn;
extendedsharedata : P SharedData;
extendedrepositories : P Repository;
newdataconnectors : P DataConnector;
newtriggers : P Trigger;
newsharedmachines : P (P ControlComponent & Node);
newcall_layers, newspawn_layers, newdataconnector_layers : Layers;
success : Success;
newsystem : System @

group(subsystems, newname, newcalls, newspawns, extendedsharedata, extendedrepositories,
newdataconnectors, newtriggers, newsharedmachines, newcall_layers,
newspawn_layers,
newdataconnector_layers) = (success, newsystem) <=>
(exists potential_mismatches : P Mismatch @
  % Check for potential mismatches given the group characteristics
  potential_mismatches = check_for_mismatches(subsystems, newcalls,
  newspawns, extendedshareddata, extendedrepositories, newdataconnectors,
  newtriggers, newsharedmachines) and
  % There are some detected potential mismatches, hence the grouping needs
  % to be carefully done in an individual basis.
  potential_mismatches /= {} <=> success = FALSE and
  % There are no detected potential mismatches, thus the grouping can be done
  % with no worries.
  potential_mismatches = {} <=>
  (success = TRUE and
   newsystem.name = newname and
   newsystem.controlunit =
   {x : ControlComponent |
    exists s : System | s in subsystems @ x in s.controlunit} and
   newsystem.initialcontrolcmps =
   {x : ControlComponent |
    exists s : System | s in subsystems @ x in s.initialcontrolcmps} and
   newsystem.globalobjs =
   {x : Object |
    exists s : System | s in subsystems @ x in s.globalobjs} and
   newsystem.controlcomponents =
   {x : ControlComponent |
    exists s : System | s in subsystems @ x in s.controlcomponents} and
   newsystem.classes =
   {x : Class |
    exists s : System | s in subsystems @ x in s.classes} and
  % Extended shareddata uses existing data but w/ larger sets of sharing
  % components/objects
  (forall d : DataComponent; cset : P ControlComponent |
   (d, cset) in extendedshareddata @
   (exists oldcset : P ControlComponent; s : System @
    s in subsystems and (d, oldcset) in s.shareddata and
    oldcset subseteq cset and
    cset subseteq getallcontrolcmps(newsystem.controlcomponents,
    newsystem.globalobjs) setunion
    getallobjs(newsystem.controlcomponents,
    newsystem.globalobjs))) and
  % Extended shared data overrides subsystems' shared data
   newsystem.shareddata =
   {x : DataComponent & P ControlComponent |
    exists s : System | s in subsystems @ x in s.shareddata} += extendedshareddata and
Extended repositories uses existing data but with larger sets of sharing components/objects

(forall dset : P DataComponent; cset : P ControlComponent; extrepr : Repository |
  extrepr in extendedrepositories and
dset = extrepr.data and
cset = extrepr.accessingcmps @
(exists oldcset : P ControlComponent; s : System; rep : Repository @
  s in subsystems and
  rep in s.repositories and
  extrepr.name = rep.name and
  dset = rep.data and
  oldcset = rep.accessingcmps and
  oldcset subseteq cset and
  cset subseteq getallcontrolcmps(newsystem.controlcomponents,
  newsystem.globalobjs) setunion
  getallobjs(newsystem.controlcomponents,
  newsystem.globalobjs)) and

Extended repositories overrides subsystems’ repositories

newsystem.repositories =
  {x : Repository |
   exists s : System | s in subsystems @ x in s.repositories and
   x notin extendedrepositories} setunion
  extendedrepositories and

All components addressed in the new calls, spawns, and dataconnectors are members of the new system control components (this constraint is redundant)

{c : ControlComponent |
  (exists call : ControlConnector | call in newcalls @
   c = call.c1 or c = call.c2) or
  (exists spawn : ControlConnector | spawn in newspawns @
   c = spawn.c1 or c = spawn.c2) or
  (exists dc : DataConnector | dc in newdataconnectors @
   c = dc.c1 or c = dc.c2)) subseteq newsystem.controlcomponents

and

All new data connectors span subsystems

(forall d : DataConnector | d in newdataconnectors @
  (exists s1, s2 : subsystems @
   d.c1 in getallcontrolcmps(s1.controlcomponents,s1.globalobjs) and
   d.c2 in getallcontrolcmps(s2.controlcomponents,s2.globalobjs) and
   s1 /= s2)) and

{x : DataConnector | exists s : System |
  s in subsystems @ x in s.dataconnectors} setint newdataconnectors = {}

and

newsystem.dataconnectors =
  {x : DataConnector |
   exists s : System | s in subsystems @ x in s.dataconnectors} setunion newdataconnectors and

All new calls span subsystems

(forall c : Call | c in newcalls @
(exists s1, s2 : subsystems @
c.c1 in getallcontrolcmps(s1.controlcomponents,s1.globalobjs) and
c.c2 in getallcontrolcmps(s2.controlcomponents,s2.globalobjs) and
s1 /= s2)) and
{x : ControlConnector | exists s : System |
s in subsystems @ x in s.calls} setint newcalls = {} and
newsystem.calls =
{x : ControlConnector |
exists s : System | s in subsystems @ x in s.calls} setunion newcalls and
% All new spawns span subsystems
(forall sp : Spawn | sp in newspawns @
(exists s1, s2 : subsystems @
sp.c1 in getallcontrolcmps(s1.controlcomponents,s1.globalobjs)
and
sp.c2 in getallcontrolcmps(s2.controlcomponents,s2.globalobjs)
and
s1 /= s2)) and
{x : ControlConnector | exists s : System |
s in subsystems @ x in s.spawns} setint newspawns = {} and
newsystem.spawns =
{x : ControlConnector |
exists s : System | s in subsystems @ x in s.spawns} setunion newspawns and
newsystem.recognizedmsgs =
{x : DataComponent |
exists s : System | s in subsystems @ x in s.recognizedmsgs} and
newsystem.events =
{x : Event |
exists s : System | s in subsystems @ x in s.events} setunion dom newtriggers and
newsystem.actions =
{x : Action |
exists s : System | s in subsystems @ x in s.actions} setunion ran newtriggers and
newsystem.triggers =
{x : Trigger |
exists s : System | s in subsystems @ x in s.triggers} setunion newtriggers and
% All layers drawn from the control components
(forall layer : Layer | layer in newcall_layers or layer in newspawn_layers or
layer in newdataconnector_layers @
first layer setunion second layer subseteq getallcontrolcmps(newsystem.controlcomponents, newsystem.globalobjs)) and
newsystem.call_layers =
{x : Layer |
exists s : System | s in subsystems @ x in s.call_layers} setunion newcall_layers and
newsystem.spawn_layers =
{x : Layer |
  exists s : System | s in subsystems
  @ x in s.spawn_layers} setunion newspawn_layers and
newsystem.dataconnector_layers =
{x : Layer |
  exists s : System | s in subsystems
  @ x in s.dataconnector_layers} setunion
newdataconnector_layers and

% Calls constrained to between layers or not constrained at all
(forall call : ControlConnector | call in newcalls @
  ((exists layer : Layer | layer in newsystem.call_layers @
    (call.c1 in first layer and call.c2 in second layer) or
    (call.c2 in first layer and call.c1 in second layer))
  or
  (forall layer : Layer | layer in newsystem.call_layers @
    {call.c1, call.c2} setint second layer = {()})
)
and

% Spawns constrained to between layers or not constrained at all
(forall spawn : ControlConnector | spawn in newspawns @
  ((exists layer : Layer | layer in newsystem.spawn_layers @
    (spawn.c1 in first layer and spawn.c2 in second layer) or
    (spawn.c2 in first layer and spawn.c1 in second layer))
  or
  (forall layer : Layer | layer in newsystem.spawn_layers @
    {spawn.c1, spawn.c2} setint second layer = {()})
)
and

% Dataconnectors constrained to between layers or not constrained at all
(forall d : DataConnector | d in newdataconnectors @
  ((exists layer : Layer | layer in newsystem.dataconnector_layers @
    (d.c1 in first layer and d.c2 in second layer) or
    (d.c2 in first layer and d.c1 in second layer))
  or
  (forall layer : Layer | layer in newsystem.dataconnector_layers @
    {d.c1, d.c2} setint second layer = {()})
)
and
newsystem.response_time = predictable <=>
(exists s : System | s in subsystems @
  s.response_time = predictable) and
newsystem.response_time = bounded <=>
(exists s : System | s in subsystems @
  s.response_time = bounded) and
not (exists s : System | s in subsystems @
  s.response_time = predictable)) and
(forall s : subsystems @ s.response_time = unbounded) <=>
(newsystem.response_time = unbounded or newsystem.response_time = cyclic) and

% If the system is cyclic it means that some subset of its control compo-
  nents form a ring with respect to its control connectors.
newsystem.response_time = cyclic <=>
(exists c_set : P ControlComponent; cn_set : P ControlConnector |
  c_set subset newsystem.controlcomponents and
cn_set subset newsystem.controlconnectors @
uring c_set = {a,b : ControlComponent |
  exists cn : ControlConnector |
\[
\begin{align*}
\text{cn in cn_set @} \\
\text{cn.c1 = a and} \\
\text{cn.c2 = b)) and} \\
\text{newsystem.backtracking = TRUE } \iff \exists s : \text{System} | s \text{ in subsystems @ s.backtracking = TRUE} \land \\
\text{newsystem.backtracking = FALSE } \iff \neg \exists s : \text{System} | s \text{ in subsystems @ s.backtracking = TRUE} \land \\
\#\text{newsystem.reconfiguration} = 1 \lor \#\text{newsystem.reconfiguration} = 2 \land \\
\text{on_the_fly_garbage_collection in newsystem.reconfiguration} \iff \exists s : \text{System} | s \text{ in subsystems @ on_the_fly_garbage_collection in s.reconfiguration} \land \\
\text{(online in newsystem.reconfiguration or offline in newsystem.reconfiguration) and} \\
\text{online in newsystem.reconfiguration} \iff \exists s : \text{System} | s \text{ in subsystems @ online in s.reconfiguration} \land \\
\text{newsystem.synchronization} = \{ x : \text{SynchronizationMechanism} | \exists s : \text{System} | s \text{ in subsystems @ x in s.synchronization} \} \land \\
\text{newsystem.nodes} = \{ x : \text{Node} | \exists s : \text{System} | s \text{ in subsystems @ x in s.nodes} \} \cup \text{ran newsharedmachines and} \\
\text{newsystem.resources} = \text{addResources} \{ x : \text{Resources} | \exists s : \text{System} | s \text{ in subsystems @ x = s.resources} \})
\end{align*}
\]

end axiom

% Starts the system as having all the initialcontrolcmps running, no control components suspended, and all others inactive.

\begin{align*}
\text{schema StartSystem} \\
\text{Delta System;}
\end{align*}

where

\begin{align*}
\text{runningcontrolcmps'} = \text{initialcontrolcmps} \land \\
\text{suspendedcontrolcmps'} = \{ \} \land \\
\text{inactivecontrolcmps'} = \text{controlcomponents setminus initialcontrolcmps};
\end{align*}

end schema

% Reflects the effects of control transfers within a system by updating the set of its running, inactive, or suspended control components.

\begin{align*}
\text{schema ActualControlTransfer} \\
\text{Delta System;}
\end{align*}

where

\begin{align*}
\% \text{ We are actually dealing with a control transfer that is part of the system.}
\end{align*}
cn in controlconnectors and
% c1 was originally running

cn.c1 in runningcontrolcmps and
% If the connector is a call or a spawn, then c2 must have been inactive, and will
% now start to run, OR it must be reentrant, and will now have another invoca-
% tion starting to run.

((cn in calls or cn in spawn
(((cn in calls or cn in spawns) =>
  (((cn.c2 in inactivecontrolcmps and
     inactivecontrolcmps' = inactivecontrolcmps setminus {cn.c2}) or
  cn.c2.reentrant = TRUE) and
  runningcontrolcmps' = runningcontrolcmps setunion {cn.c2} and
% If c2 is an applet, then node becomes that of the caller.
  cn.c2.applet = TRUE => cn.c2.nodes = cn.c1.nodes and
% If the connector is a call, then c1 will be suspended until c2 terminates.
  cn in calls => (runningcontrolcmps' = runningcontrolcmps setminus {cn.c1} and
    suspendedcontrolcmps' = suspendedcontrolcmps setunion {cn.c1}) and
% If the connector is a return, then c1 will stop running and become inactive.
  (cn in returns =>
    (cn.c1 in runningcontrolcmps and
     runningcontrolcmps' = runningcontrolcmps setminus {cn.c1} and
     inactivecontrolcmps' = inactivecontrolcmps setunion {cn.c1} and
% Returning from a call will make c2 start running again.
     (exists call : Call @ (call, cn) in callreturns) =>
       (runningcontrolcmps' = runningcontrolcmps setunion {cn.c2} and
        suspendedcontrolcmps' = suspendedcontrolcmps setminus {cn.c2}) and
% If the connector is a swap-in, then c2 will resume running.
      (cn in swap_ins =>
        (cn.c2 in suspendedcontrolcmps and
         suspendedcontrolcmps' = suspendedcontrolcmps setminus {cn.c2} and
         runningcontrolcmps' = runningcontrolcmps setunion {cn.c2}) and
% If the connector is an interrupt or a terminate, then c2 must have not been inac-
% tive
 ((cn in interrupts or cn in terminates) =>
    (cn.c2notin inactivecontrolcmps and
     cn.c2 in runningcontrolcmps =>
      runningcontrolcmps' = runningcontrolcmps setminus {cn.c2} and
     cn.c2 in suspendedcontrolcmps and cn in terminates =>
      suspendedcontrolcmps' = suspendedcontrolcmps setminus {cn.c2} and
% If the connector is an interrupt, then c2 will be suspended.
      cn in interrupts => suspendedcontrolcmps' = suspendedcontrolcmps setunion
      {cn.c2} and
% If the connector is a terminate, then c2 will become inactive.
      cn in terminates => inactivecontrolcmps' = inactivecontrolcmps setunion
      {cn.c2}))));
end schema

% Confirm that the actual data transfer can occur at that given point in time.
schema ActualDataTransfer
  s : System;
  dcn : DataConnector;
data : DataComponent;

where
  % We are actually dealing with a data transfer that is part of the system.
dcn in s.dataconnectors and
data in s.recognizedmsgs and
  % c1 was originally running
dcn.c1 in s.runningcontrolcmps and
  % Data can be sent by c1 to c2 through dcn (the part on c1 and c2 is checked
  when defining dcn).
(forall t : DataType | t in ran(data.type) @ t in dcn.dataformat);
  % If c2 is not running then dcn = buffered or dcn /= blocking otherwise either data is
  lost or dcn just hangs there indefinitely.
dcn.c2 notin s.runningcontrolcmps =>
  (dcn.p1.blocking = FALSE or
   dcn.buffersize /= 0);

end schema

end specification
Appendix D: Time Related Classification of Systems and Operations

Systems’ response times can be classified as: predicted, bounded, or unbounded. The ones classified as unbounded may be either scheduled (swapped in and out as appropriate until termination), or cyclic, where cyclic can still have each cycle be predicted, bounded, or unbounded (scheduled).

Operations can also be classified with respect to time, yielding: sequential ones (invokes, followed by, and interrupted by), and parallel ones (spawns, triggers, together cooperative, and together independent).

Based on the classifications above we can define the expected results of applying various operations with respect to time. Here we have only covered the sequential time operations.
## Involves

<table>
<thead>
<tr>
<th>invokes</th>
<th>P(b)</th>
<th>B(b)</th>
<th>S</th>
<th>C[P(b)]</th>
<th>C[B(b)]</th>
<th>C[S]</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(a)</td>
<td>P(a+b+2swaps)</td>
<td>B(a+b+2swaps)</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>B(a)</td>
<td>B(a+b+2swaps)</td>
<td>B(a+b+2swaps)</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[P(a)]</td>
<td>C[P(a) invokes P(b)]</td>
<td>C[P(a) invokes B(b)]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[B(a)]</td>
<td>C[B(a) invokes P(b)]</td>
<td>C[B(a) invokes B(b)]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[S]</td>
<td>C[S invokes P(b)]</td>
<td>C[S invokes B(b)]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table 12: Operation Involves**

## Followed_By

<table>
<thead>
<tr>
<th>followed_by</th>
<th>P(b)</th>
<th>B(b)</th>
<th>S</th>
<th>C[P(b)]</th>
<th>C[B(b)]</th>
<th>C[S]</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(a)</td>
<td>P(a+b)</td>
<td>B(a+b)</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>B(a)</td>
<td>B(a+b)</td>
<td>B(a+b)</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>C[P(a)]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[B(a)]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[S]</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table 13: Operation Followed_By**
**Interrupted_By**

<table>
<thead>
<tr>
<th>interru_by</th>
<th>P(b)</th>
<th>B(b)</th>
<th>S</th>
<th>C[P(b)]</th>
<th>C[B(b)]</th>
<th>C[S]</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(a)</td>
<td>P(a+b+2swaps)</td>
<td>B(a+b+2swaps)</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>B(a)</td>
<td>B(a+b+2swaps)</td>
<td>B(a+b+2swaps)</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[P(a)]</td>
<td>C[P(a) interru_by P(b)]</td>
<td>C[P(a) interru_by B(b)]</td>
<td>C[P(a) interru_by S]</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[B(a)]</td>
<td>C[B(a) interru_by P(b)]</td>
<td>C[B(a) interru_by B(b)]</td>
<td>C[B(a) interru_by S]</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>C[S]</td>
<td>C[S interru_by P(b)]</td>
<td>C[S interru_by B(b)]</td>
<td>C[S interru_by S]</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Table 14: Operation Interrupted_By**

**Generalizations from time sequential operations:**

- $C[r]$ followed_by $t$ (where $r$ in $\{P(a), B(a), S\}$ and $t$ in $\{P(b), B(b), S, C[P(b)], C[B(b)], C[S]\}$) **XX** b is unreachable

- $r$ invokes $C[t]$ (where $r$ in $\{P(a), B(a), S, C[P(a)], C[B(a)], C[S]\}$ and $t$ in $\{P(b), B(b), S\}$) **XX** a deadlocks

- $r$ interrupted_by $C[t]$ (where $r$ in $\{P(a), B(a), S, C[P(a)], C[B(a)], C[S]\}$ and $t$ in $\{P(b), B(b), S\}$) **XX** a deadlocks

- $C[r]$ invokes $t$ (where $r$ in $\{P(a), B(a), S\}$ and $t$ in $\{P(b), B(b), S\}$) = $C[r$ invokes $t]$

- $C[r]$ interrupted_by $t$ (where $r$ in $\{P(a), B(a), S\}$ and $t$ in $\{P(b), B(b), S\}$) = $C[r$ interrupted_by $t]$

- $S\ OP\ r$ (where $OP$ in $\{invokes, followed_by, interrupted_by\}$ and $r$ in $\{P(b), B(b), S\}$) = $S$

- $r\ OP\ S$ (where $r$ in $\{P(a), B(a)\}$ and $OP$ in $\{invokes, followed_by, interrupted_by\}$) = $S$

- $r$ invokes $B(b)$ (where $r$ in $\{P(a), B(a)\}$) = $B(a+b+2swaps)$
r followed_by B(b) (where r in \{P(a), B(a)\}) = B(a+b)

r interrupted_by B(b) (where r in \{P(a), B(a)\}) = B(a+b+2\text{swaps})
Appendix E: AAA Mismatch Analysis Results for Real-Time vs. All Other Styles

Real-Time with Blackboard

AAA Mismatch Analysis Results
-----------------------------

System name: RTwithBB
Date: 9/3/98
Time: 12:49:57

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and BBsub

-----------------------------------------

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and BBsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and BBsub (data layering unknown) (on data layer)

-----------------------------------------

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.
Violating subsystems
RTsub, BBsub

-----------------------------------------

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and BBsub (threads initiating unknown)

-----------------------------------------

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and BBsub
BBsub and RTsub

-----------------------------------------

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub and BBsub

-----------------------------------------

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub and BBsub

-----------------------------------------

Short description
A (triggered) spawn is made into or out of a subsystem which is not concurrent.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and BBsub (concurrency unknown)

-------------------------------

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, BBsub

-------------------------------

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, BBsub

-------------------------------

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and BBsub

-------------------------------

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.
Violating subsystems
RTsub and BBsub

-----------------------------------------

Short description
Erroneous assumption of single threaded.

Problem
Synchronization problems on accessing shared data, and/or resource contention.

Violating subsystems
RTsub and BBsub (concurrency unknown)

-----------------------------------------

Short description
(Triggered) Call to a cyclic (non-terminating) subsystem.

Problem
Control will never return to the caller.

Violating subsystems
RTsub and BBsub (response time unknown)

-----------------------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, BBsub

-----------------------------------------

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and BBsub (encapsulation unknown)
BBsub and RTsub (encapsulation unknown)

-----------------------------------------

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and BBsub (encapsulation unknown)
BBsub and RTsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and BBsub (encapsulation unknown)

-----------------------------------------

Short description
More than one central control unit exists.

Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTsub, BBsub

-----------------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and BBsub
BBsub and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.
Violating subsystems
RTsub and BBsub
BBsub and RTsub

-----------------------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, BBsub

-----------------------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------

Short description
(Triggered) Call or spawn from a subsystem that may later backtrack.

Problem
While backtracking may cause undesired side effects on the called/spawned end.

Violating subsystems
BBsub and RTsub

-----------------------------------------

Short description
Data being transferred from some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and BBsub

-----------------------------------------
Short description
Sharing data with some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and BBsub

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and BBsub
BBsub (response times unknown) and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and BBsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, BBsub

-----------------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.
Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and BBsub (concurrency unknown)

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and BBsub

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and BBsub

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and BBsub (on the fly garbage collection unknown)

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and BBsub (encapsulation unknown)
Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and BBsub (encapsulation unknown)

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and BBsub

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, BBsub

Short description
Potential database and/or DBMS heterogeneity problems may occur.

Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
BBsub, RTsub (shared repository unknown)

Short description
Inaccessible shared data.
Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and BBsub (downloadable applets unknown)

-----------------------------------------

Real-Time with Closed-Loop Feedback Control

AAA Mismatch Analysis Results

----------

System name: RTwithCLFC
Date: 9/3/98
Time: 13:02:43

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and CLFCsub

-----------------------------------------

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and CLFCsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and CLFCsub (data layering unknown) (on data layer)

-----------------------------------------
Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, CLFCsub

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and CLFCsub (threads initiating unknown)

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and CLFCsub
CLFCsub and RTsub

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub (data connectors unknown) and CLFCsub (data connectors unknown)

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.
Violating subsystems
RTsub (shared data variables unknown) and CLFCsub

Short description
A (triggered) spawn is made into or out of a subsystem which is not concurrent.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and CLFCsub (concurrency unknown)

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, CLFCsub

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, CLFCsub

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and CLFCsub
Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and CLFCsub

-------------------------------

Short description
Erroneous assumption of single threaded.

Problem
Synchronization problems on accessing shared data, and/or resource contention.

Violating subsystems
RTsub and CLFCsub (concurrency unknown)

-------------------------------

Short description
(Triggered) Call to a cyclic (non-terminating) subsystem.

Problem
Control will never return to the caller.

Violating subsystems
RTsub and CLFCsub (response time unknown)

-------------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, CLFCsub

-------------------------------

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and CLFCsub (encapsulation unknown)
CLFCsub and RTsub (encapsulation unknown)

-----------------------------------------

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and CLFCsub (encapsulation unknown)
CLFCsub and RTsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and CLFCsub (encapsulation unknown)

-----------------------------------------

Short description
More than one central control unit exists.

Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTsub, CLFCsub (control unit unknown)

-----------------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and CLFCsub
CLFCsub and RTsub
Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and CLFCsub
CLFCsub and RTsub

-----------------------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, CLFCsub (component priorities unknown)

-----------------------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and CLFCsub
CLFCsub (response times unknown) and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to
some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and CLFCsub

-------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, CLFCsub

-------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and CLFCsub (concurrency unknown)

-------------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and CLFCsub (on the fly garbage collection unknown)

-------------------------------

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a
data transfer.

Violating subsystems
RTsub (encapsulation unknown) and CLFCsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and CLFCsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and CLFCsub

-----------------------------------------

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, CLFCsub

-----------------------------------------

Short description
Potential database and/or DBMS heterogeneity problems may occur.

Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
RTsub (shared repository unknown), CLFCsub (shared repository unknown)
Real-Time with Database-Centric

AAA Mismatch Analysis Results

System name: RTwithDB

Date: 9/3/98
Time: 13:02:49

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and DBsub

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and DBsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and DBsub (data layering unknown) (on data layer)

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, DBsub

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and DBsub (threads initiating unknown)

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and DBsub
DBsub and RTsub

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub (data connectors unknown) and DBsub (data connectors unknown)

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub and DBsub

Short description
A node resource is overused.
Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, DBsub

-----------------------------

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, DBsub

-----------------------------

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and DBsub

-----------------------------

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and DBsub

-----------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, DBsub
Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and DBsub (encapsulation unknown)
DBsub and RTsub (encapsulation unknown)

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and DBsub (encapsulation unknown)
DBsub and RTsub (encapsulation unknown)

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and DBsub (encapsulation unknown)

Short description
More than one central control unit exists.

Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTsub, DBsub

Short description
Sharing data with a reentrant component.
Problem
Sharing of data may occur with the incorrect invocation of a component.

Violating subsystems
RTsub and DBsub

Short description
A reentrant component is either sending or receiving a data transfer.

Problem
Potential incorrect assumption of which invocation of a component is either sending or receiving a data transfer.

Violating subsystems
RTsub and DBsub

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
DBsub and RTsub

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
DBsub and RTsub

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, DBsub (component priorities unknown)
Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

Short description
(Triggered) Call or spawn from a subsystem that may later backtrack.

Problem
While backtracking may cause undesired side effects on the called/spawned end.

Violating subsystems
DBsub and RTsub

Short description
Data being transferred from some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and DBsub

Short description
Sharing data with some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and DBsub

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.
Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and DBsub

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and DBsub

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, DBsub

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and DBsub

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and DBsub (preemption unknown)

-----------------------------------------
Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and DBsub (preemption unknown)

-----------------------------------------
Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and DBsub (on the fly garbage collection unknown)

-----------------------------------------
Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and DBsub (encapsulation unknown)

-----------------------------------------
Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and DBsub (encapsulation unknown)
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and DBsub

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, DBsub

Short description
Potential database and/or DBMS heterogeneity problems may occur.

Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
DBsub, RTsub (shared repository unknown)

Short description
Inaccessible shared data.

Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and DBsub (downloadable applets unknown)
AAA Mismatch Analysis Results

System name: RTwithDP

Date: 9/3/98
Time: 13:02:55

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and DPsub

-----------------------------------------

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and DPsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and DPsub (data layering unknown) (on data layer)

-----------------------------------------

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, DPsub (triggering unknown)
Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and DPsub
DPsub and RTsub

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub (data connectors unknown) and DPsub

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub and DPsub

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, DPsub

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.
Violating subsystems
RTsub, DPsub (triggering unknown)

-----------------------------------------
Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and DPsub

-----------------------------------------
Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and DPsub

-----------------------------------------
Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, DPsub

-----------------------------------------
Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and DPsub (encapsulation unknown)
DPsub and RTsub (encapsulation unknown)
Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and DPsub (encapsulation unknown)
DPsub and RTsub (encapsulation unknown)

-----------------------------

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and DPsub (encapsulation unknown)

-----------------------------

Short description
Sharing data with a reentrant component.

Problem
Sharing of data may occur with the incorrect invocation of a component.

Violating subsystems
RTsub and DPsub (reentrance unknown)

-----------------------------

Short description
A reentrant component is either sending or receiving a data transfer.

Problem
Potential incorrect assumption of which invocation of a component is either sending or receiving a data transfer.

Violating subsystems
RTsub and DPsub (reentrance unknown)

-----------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.
Violating subsystems
RTsub and DPsub (reentrance unknown)
DPsub and RTsub

==============================================

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and DPsub (reentrance unknown)
DPsub and RTsub

==============================================

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub

==============================================

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

==============================================

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and DPsub
Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and DPsub

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, DPsub

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and DPsub

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and DPsub (preemption unknown)

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.
Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and DPsub (preemption unknown)

-----------------------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and DPsub (on the fly garbage collection unknown)

-----------------------------------------

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and DPsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and DPsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and DPsub
Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, DPsub

Short description
Inaccessible shared data.

Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and DPsub (downloadable applets unknown)

Real-Time with Event-Based

AAA Mismatch Analysis Results

System name: RTwithEB
Date: 9/3/98
Time: 13:03:02

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and EBsub

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and EBsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and EBsub (data layering unknown) (on data layer)

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, EBsub

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and EBsub (threads initiating unknown)

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and EBsub
EBsub and RTsub
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub (data connectors unknown) and EBsub (data connectors unknown)

------------------------------------------------------------------------

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub (shared data variables unknown) and EBsub (shared data variables unknown)

------------------------------------------------------------------------

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, EBsub

------------------------------------------------------------------------

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, EBsub

------------------------------------------------------------------------

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.
Violating subsystems
RTsub and EBsub

-------------------------------

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and EBsub

-------------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, EBsub

-------------------------------

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and EBsub (encapsulation unknown)
EBsub and RTsub (encapsulation unknown)

-------------------------------

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and EBsub (encapsulation unknown)
EBsub and RTsub (encapsulation unknown)

-------------------------------
Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and EBsub (encapsulation unknown)

-----------------------------------------

Short description
More than one central control unit exists.

Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTsub, EBsub (control unit unknown)

-----------------------------------------

Short description
Sharing data with a reentrant component.

Problem
Sharing of data may occur with the incorrect invocation of a component.

Violating subsystems
RTsub and EBsub (reentrance unknown)

-----------------------------------------

Short description
A reentrant component is either sending or receiving a data transfer.

Problem
Potential incorrect assumption of which invocation of a component is either sending or receiving a data transfer.

Violating subsystems
RTsub and EBsub (reentrance unknown)

-----------------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.
Violating subsystems
RTsub and EBsub (reentrance unknown)
EBsub and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and EBsub (reentrance unknown)
EBsub and RTsub

-----------------------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub

-----------------------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and EBsub
Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and EBsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, EBsub

-----------------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and EBsub

-----------------------------------------

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and EBsub (preemption unknown)

-----------------------------------------

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.
Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and EBsub (preemption unknown)

-----------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and EBsub (on the fly garbage collection unknown)

-----------------------------

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and EBsub (encapsulation unknown)

-----------------------------

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and EBsub (encapsulation unknown)

-----------------------------

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and EBsub
Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, EBsub

Short description
Potential database and/or DBMS heterogeneity problems may occur.

Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
RTsub (shared repository unknown), EBsub (shared repository unknown)

Short description
Inaccessible shared data.

Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and EBsub (downloadable applets unknown)

Real-Time with Internet Distributed Entities

AAA Mismatch Analysis Results

System name: RTwithIDE

Date: 9/3/98
Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and IDEsub

-----------------------------------------

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub and IDEsub (on control layer)
RTsub (data layering unknown) and IDEsub (data layering unknown) (on data layer)

-----------------------------------------

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, IDEsub

-----------------------------------------

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and IDEsub
IDEsub and RTsub

-----------------------------------------
Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub (data connectors unknown) and IDEsub

---------------------------------------------------------------

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub (shared data variables unknown) and IDEsub (shared data variables unknown)

---------------------------------------------------------------

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, IDEsub

---------------------------------------------------------------

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and
if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, IDEsub

---------------------------------------------------------------

Short description
Data connectors connecting control components that are not always active may lead
into deadlock.

Problem
Possibility of deadlock on the control component sending the data.
Violating subsystems
RTsub and IDEsub

-----------------------------------------
Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and IDEsub

-----------------------------------------
Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, IDEsub

-----------------------------------------
Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and IDEsub (encapsulation unknown)
IDEsub and RTsub (encapsulation unknown)

-----------------------------------------
Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and IDEsub (encapsulation unknown)
IDEsub and RTsub (encapsulation unknown)
Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and IDEsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing data with a reentrant component.

Problem
Sharing of data may occur with the incorrect invocation of a component.

Violating subsystems
RTsub and IDEsub

-----------------------------------------

Short description
A reentrant component is either sending or receiving a data transfer.

Problem
Potential incorrect assumption of which invocation of a component is either sending or receiving a data transfer.

Violating subsystems
RTsub and IDEsub

-----------------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
IDEsub and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.
Violating subsystems
IDEsub and RTsub

-----------------------------------------
Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, IDEsub (component priorities unknown)

-----------------------------------------
Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------
Short description
(Triggered) Call or spawn from a subsystem that may later backtrack.

Problem
While backtracking may cause undesired side effects on the called/spawned end.

Violating subsystems
IDEsub (backtracking unknown) and RTsub

-----------------------------------------
Short description
Data being transferred from some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and IDEsub (backtracking unknown)
Short description
Sharing data with some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and IDEsub (backtracking unknown)

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and IDEsub

-----------------------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and IDEsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, IDEsub

-----------------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.
Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and IDEsub

---------------------------------------------------------------

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and IDEsub (preemption unknown)

---------------------------------------------------------------

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and IDEsub (preemption unknown)

---------------------------------------------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and IDEsub

---------------------------------------------------------------

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and IDEsub (encapsulation unknown)
Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and IDEsub (encapsulation unknown)

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and IDEsub

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, IDEsub

Short description
Potential database and/or DBMS heterogeneity problems may occur.

Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
RTsub (shared repository unknown), IDEsub (shared repository unknown)

Short description
Inaccessible shared data.
Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and IDEsub

Real-Time with Logic Programming

AAA Mismatch Analysis Results

System name: RTwithLP
Date: 9/3/98
Time: 13:03:14

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and LPsub

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and LPsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and LPsub (data layering unknown) (on data layer)
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, LPsub

-------------------------------

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and LPsub (threads initiating unknown)

-------------------------------

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and LPsub
LPsub and RTsub

-------------------------------

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub and LPsub

-------------------------------

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub and LPsub

Short description
A (triggered) spawn is made into or out of a subsystem which is not concurrent.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and LPsub (concurrency unknown)

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, LPsub

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, LPsub

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and LPsub

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and LPsub

-----------------------------

Short description
Erroneous assumption of single threaded.

Problem
Synchronization problems on accessing shared data, and/or resource contention.

Violating subsystems
RTsub and LPsub (concurrency unknown)

-----------------------------

Short description
(Triggered) Call to a cyclic (non-terminating) subsystem.

Problem
Control will never return to the caller.

Violating subsystems
RTsub and LPsub (response time unknown)

-----------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, LPsub

-----------------------------

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and LPsub (encapsulation unknown)
LPSUB and RTSUB (encapsulation unknown)

-------------------------------

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTSUB and LPSUB (encapsulation unknown)
LPSUB and RTSUB (encapsulation unknown)

-------------------------------

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTSUB (encapsulation unknown) and LPSUB (encapsulation unknown)

-------------------------------

Short description
More than one central control unit exists.

Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTSUB, LPSUB

-------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTSUB and LPSUB
LPSUB and RTSUB

-------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and LPsub
LPsub and RTsub

-----------------------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, LPsub

-----------------------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------

Short description
(Triggered) Call or spawn from a subsystem that may later backtrack.

Problem
While backtracking may cause undesired side effects on the called/spawned end.

Violating subsystems
LPsub and RTsub

-----------------------------------------

Short description
Data being transferred from some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.
Violating subsystems
RTsub and LPsub

-----------------------------------------

Short description
Sharing data with some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and LPsub

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and LPsub
LPsub (response times unknown) and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and LPsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, LPsub
Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and LPsub (concurrency unknown)

-----------------------------

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and LPsub

-----------------------------

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and LPsub

-----------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and LPsub (on the fly garbage collection unknown)

-----------------------------

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.
Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and LPsub (encapsulation unknown)

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and LPsub (encapsulation unknown)

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and LPsub

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, LPsub

Short description
Potential database and/or DBMS heterogeneity problems may occur.

Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.
Violating subsystems
LPsub, RTsub (shared repository unknown)

-----------------------------------------

Short description
Inaccessible shared data.

Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and LPsub (downloadable applets unknown)

-----------------------------------------

Real-Time with Main-Subroutine

AAA Mismatch Analysis Results
-------------------------------

System name: RTwithMS
Date: 9/3/98
Time: 13:03:21

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and MSsub

-----------------------------------------

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints
Violating subsystems
RTsub (control layering unknown) and MSsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and MSsub (data layering unknown) (on data layer)

-----------------------------------------

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and MSsub

-----------------------------------------

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and MSsub
MSsub and RTsub

-----------------------------------------

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub and MSsub

-----------------------------------------

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub (shared data variables unknown) and MSsub
Short description
A (triggered) spawn is made into or out of a subsystem which is not concurrent.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and MSsub

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, MSsub

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and MSsub

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and MSsub

Short description
Erroneous assumption of single threaded.

Problem
Synchronization problems on accessing shared data, and/or resource contention.
Violating subsystems
RTsub and MSsub

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, MSsub

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and MSsub (encapsulation unknown)
MSsub and RTsub (encapsulation unknown)

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and MSsub (encapsulation unknown)
MSsub and RTsub (encapsulation unknown)

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and MSsub (encapsulation unknown)
Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and MSsub
MSsub and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and MSsub
MSsub and RTsub

-----------------------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub

-----------------------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.
Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and MSsub

-----------------------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and MSsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, MSsub

-----------------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and MSsub

-----------------------------------------

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and MSsub

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and MSsub

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and MSsub (on the fly garbage collection unknown)

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and MSsub (encapsulation unknown)

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and MSsub (encapsulation unknown)
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and MSsub

-----------------------------

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, MSsub

-----------------------------

Real-Time with Pipe-and-Filter

AAA Mismatch Analysis Results
-----------------------------

System name: RTwithPF
Date: 9/3/98
Time: 13:03:28

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and PFsub

-----------------------------
Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RSub (control layering unknown) and PFSub (control layering unknown) (on control layer)
RSub (data layering unknown) and PFSub (data layering unknown) (on data layer)

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RSub and PFSub

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RSub and PFSub
PFSub and RSub

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RSub (data connectors unknown) and PFSub

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.
Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, PFsub

-----------------------------------------

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, PFsub

-----------------------------------------

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and PFsub (encapsulation unknown)
PFsub and RTsub (encapsulation unknown)

-----------------------------------------

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and PFsub (encapsulation unknown)
PFsub and RTsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and PFsub (encapsulation unknown)

-----------------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and PFsub
PFsub and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.
Violating subsystems
RTsub and PFsub
PFsub and RTsub

-----------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub

-----------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and PFsub

-----------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, PFsub

-----------------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and PFsub

-----------------------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.
Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and PFsub (on the fly garbage collection unknown)

Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and PFsub (encapsulation unknown)

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and PFsub (encapsulation unknown)

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and PFsub

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, PFsub

-----------------------------------------
Short description
Inaccessible shared data.

Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and PFsub (downloadable applets unknown)

-----------------------------------------

Real-Time with Rule-Based

AAA Mismatch Analysis Results

System name: RTwithRB
Date: 9/3/98
Time: 13:03:34

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints
Violating subsystems
RTsub (control layering unknown) and RBsub (control layering unknown) (on control layer)
RTsub (data layering unknown) and RBsub (data layering unknown) (on data layer)

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, RBsub

Short description
A (triggered) spawn is made into or out of a subsystem which originally forbade them.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and RBsub (threads initiating unknown)

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and RBsub
RBsub and RTsub

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub and RBsub
Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
A (triggered) spawn is made into or out of a subsystem which is not concurrent.

Problem
May cause synchronization problems, as well as resources contention.

Violating subsystems
RTsub and RBsub (concurrency unknown)

-----------------------------------------

Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, RBsub

-----------------------------------------

Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, RBsub

-----------------------------------------

Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.
Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
Erroneous assumption of single threaded.

Problem
Synchronization problems on accessing shared data, and/or resource contention.

Violating subsystems
RTsub and RBsub (concurrency unknown)

-----------------------------------------

Short description
(Triggered) Call to a cyclic (non-terminating) subsystem.

Problem
Control will never return to the caller.

Violating subsystems
RTsub and RBsub (response time unknown)

-----------------------------------------

Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, RBsub

-----------------------------------------

Short description
(Triggered) Call to a private method.
Problem
Method not accessible to the caller.

Violating subsystems
RTsub and RBsub (encapsulation unknown)
RBsub and RTsub (encapsulation unknown)

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and RBsub (encapsulation unknown)
RBsub and RTsub (encapsulation unknown)

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and RBsub (encapsulation unknown)

Short description
More than one central control unit exists.

Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTsub, RBsub

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and RBsub
RBsub and RTsub

-----------------------------------------
Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and RBsub
RBsub and RTsub

-----------------------------------------
Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, RBsub

-----------------------------------------
Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.

Violating subsystems
RTsub

-----------------------------------------
Short description
(Triggered) Call or spawn from a subsystem that may later backtrack.

Problem
While backtracking may cause undesired side effects on the called/spawned end.

Violating subsystems
RBsub and RTsub
Short description
Data being transferred from some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
Sharing data with some component(s) that may later backtrack.

Problem
Backtracking may cause undesired side effects on the overall composed system state.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and RBsub
RBsub (response times unknown) and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) no originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.
Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, RBsub

-----------------------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and RBsub (concurrency unknown)

-----------------------------------------

Short description
(Triggered) Call to a component that should be preemptable and isn’t.

Problem
Callee will not be preempted in situations when it should.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
(Triggered) Spawn to a component that should be preemptable and isn’t.

Problem
Spawnee will not be preempted in situations when it should.

Violating subsystems
RTsub and RBsub

-----------------------------------------

Short description
(Triggered) Call to a component that performs on the fly garbage collection.

Problem
Undesirable side effects on the overall predictable response times.

Violating subsystems
RTsub and RBsub (on the fly garbage collection unknown)
Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and RBsub (encapsulation unknown)

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and RBsub (encapsulation unknown)

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and RBsub

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, RBsub

Short description
Potential database and/or DBMS heterogeneity problems may occur.
Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
RBsub, RTsub (shared repository unknown)

-------------------------------

Short description
Inaccessible shared data.

Problem
Applets have no access to the local file system of the client side (i.e., the applet cannot really access the data that is to be shared).

Violating subsystems
RTsub and RBsub (downloadable applets unknown)

-------------------------------

**Real-Time with Real-Time**

AAA Mismatch Analysis Results

System name: RTwithRT2

Date: 9/3/98
Time: 13:03:41

Mismatches:

Short description
Two concurrent threads share data, with potential synchronization problems.

Problem
Potential synchronization problems for concurrent threads while sharing data.

Violating subsystems
RTsub and RTsub2
Short description
A layering constraint is violated.

Problem
Bridging connector may ignore existing layering constraints

Violating subsystems
RTsub (control layering unknown) and RTsub2 (control layering unknown) (on control layer)
RTsub (data layering unknown) and RTsub2 (data layering unknown) (on data layer)

Short description
Different sets of recognized events are used by two subsystems that permit triggers.

Problem
A trigger may not be recognized by some subsystem that should.

Violating subsystems
RTsub, RTsub2

Short description
An unrecognized triggering event is used.

Problem
The trigger will not cause the expected behavior, it will never fire the related actions.

Violating subsystems
RTsub and RTsub2
RTsub2 and RTsub

Short description
A data connector is made into a subsystem which originally forbade them.

Problem
Unclear on how data is to be handled for sending or receiving.

Violating subsystems
RTsub (data connectors unknown) and RTsub2 (data connectors unknown)

Short description
A shared data relationship refers to a subsystem which originally forbade them.

Problem
May cause synchronization problems.

Violating subsystems
RTsub (shared data variables unknown) and RTsub2 (shared data variables unknown)

-----------------------------------------
Short description
A node resource is overused.

Problem
Resource overusage such as memory and disk space.

Violating subsystems
RTsub, RTsub2

-----------------------------------------
Short description
There is a non-deterministic set of actions that could be caused by a trigger.

Problem
It is not clear which set of actions should actually occur based on the composition, and if all, it is not clear what the ordering should be.

Violating subsystems
RTsub, RTsub2

-----------------------------------------
Short description
Data connectors connecting control components that are not always active may lead into deadlock.

Problem
Possibility of deadlock on the control component sending the data.

Violating subsystems
RTsub and RTsub2

-----------------------------------------
Short description
Data connectors connecting control components that are not always active.

Problem
Possibility of loss of data.

Violating subsystems
RTsub and RTsub2
Short description
Erroneous assumption of same underlying platform.

Problem
Cannot force fit parts to a given platform.

Violating subsystems
RTsub, RTsub2

-----------------------------------------

Short description
(Triggered) Call to a private method.

Problem
Method not accessible to the caller.

Violating subsystems
RTsub and RTsub2 (encapsulation unknown)
RTsub2 and RTsub (encapsulation unknown)

-----------------------------------------

Short description
(Triggered) Spawn to a private method.

Problem
Method not accessible to the spawner.

Violating subsystems
RTsub and RTsub2 (encapsulation unknown)
RTsub2 and RTsub (encapsulation unknown)

-----------------------------------------

Short description
Sharing private data.

Problem
Data may not be accessible to all of the sharing entities being composed.

Violating subsystems
RTsub (encapsulation unknown) and RTsub2 (encapsulation unknown)

-----------------------------------------

Short description
More than one central control unit exists.
Problem
All central control units assume they have absolute control on the execution sequencing.

Violating subsystems
RTsub, RTsub2

-----------------------------------------

Short description
(Triggered) Call to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and RTsub2
RTsub2 and RTsub

-----------------------------------------

Short description
(Triggered) Spawn to a non-reentrant component.

Problem
Component may already be running.

Violating subsystems
RTsub and RTsub2
RTsub2 and RTsub

-----------------------------------------

Short description
Composition involves one or more prioritized parts.

Problem
Not clear where a component(s) with or without an associated priority fits in an overall separately prioritized structure.

Violating subsystems
RTsub, RTsub2

-----------------------------------------

Short description
A prioritized system sharing a machine with some other system.

Problem
Unclear on how priorities compare across the various parts, which affects the way interrupts are to be applied.
Violating subsystems
RTsub

-----------------------------

Short description
(Triggered) Call from a subsystem requiring some predictable response times to some component(s) not originally considered.

Problem
Unexpected side effects on original predicted response times may occur.

Violating subsystems
RTsub and RTsub2
RTsub2 and RTsub

-----------------------------

Short description
(Triggered) Spawn to or from a subsystem requiring some predictable response times to some component(s) not originally considered.

Problem
May have side effects on original predicted response times. The original predicted response times did not account for concurrently executing with these other components.

Violating subsystems
RTsub and RTsub2

-----------------------------

Short description
Only part of the resulting system automatically reconfigures upon failure.

Problem
Only part of the resulting system automatically reconfigures upon failure.

Violating subsystems
Reconfiguration unknown: RTsub, RTsub2

-----------------------------

Short description
Some components that were expected to synchronize have different synchronization mechanisms.

Problem
Synchronization between concurrent parts cannot properly take place.

Violating subsystems
RTsub and RTsub2
Short description
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Problem
Incorrect assumption of which instantiation of an object is either sending or receiving a data transfer.

Violating subsystems
RTsub (encapsulation unknown) and RTsub2 (encapsulation unknown)

-----------------------------------------

Short description
Sharing data with the incorrect instantiation of an object.

Problem
Sharing data with the incorrect instantiation of an object.

Violating subsystems
RTsub (encapsulation unknown) and RTsub2 (encapsulation unknown)

-----------------------------------------

Short description
Sharing or transferring data with differing underlying representations.

Problem
Communications concerning the specific data will not properly occur.

Violating subsystems
RTsub and RTsub2

-----------------------------------------

Short description
Resource contention.

Problem
Predictable response time indirectly affected because there may be some resource contention not originally considered.

Violating subsystems
RTsub, RTsub2

-----------------------------------------

Short description
Potential database and/or DBMS heterogeneity problems may occur.
Problem
Problems may arise on semantic heterogeneity, differing data items granularity, databases being distributed or not, databases being replicated or not, and/or databases having different structural organizations.

Violating subsystems
RTsub (shared repository unknown), RTsub2 (shared repository unknown)

-----------------------------------------
Appendix F: Acronyms

- AAA - Architect’s Automated Assistant
- ADL - Architecture Description Language
- AI - Artificial Intelligence
- API - Application Program Interface
- CASE - Computer Aided Software Engineering
- COTS - Commercial Off The Shelf
- CPU - Central Processing Unit
- CSE - Center for Software Engineering
- DBMS - Database Management System
- DSSA - Domain Specific Software Architecture
- GOTS - Government Off The Shelf
- GUI - Graphical User Interface
- OTS - Off The Shelf
- SGS - Satellite Ground Station
- USC - University of Southern California