Abstract

This document presents examples of MontiArcView C&C views, C&C views specifications and C&C models.
# Contents

1 Introduction 2

2 Robotic Arm Example 3
2.1 C&C Views and Specification $S_1$ 3
2.2 C&C Views and Specification $S_1IC$ 13
2.3 C&C Views and Specification $S_1LC$ 17
2.4 C&C Views and Specification $S_1IMP$ 21
2.5 C&C Views and Specification $S_1CS$ 26
2.6 C&C Views and Specification $S_1HIER$ 27
2.7 C&C Views and Specification $S_2$ 30
2.8 C&C Views and Specification $S_2XCYL$ 32

3 Pump Station Example 34
3.1 C&C Views and Specification ALL 34
3.2 C&C Views and Specification ALL $-$ EMRG 45
3.3 C&C Views and Specification ALL $-$ EMRG $-$ FIX 47
3.4 C&C Views and Specification PHIS $-$ SIM 52
3.5 C&C Views and Specification PHIS $-$ SIM2 55
3.6 C&C Views and Specification PHIS $-$ SIM2 $-$ BADS 58
3.7 C&C Views and Specification PHIS $-$ SIM2 $-$ NO $-$ EMRG 59
3.8 C&C Views and Specification SENSOR $-$ LIB 63

4 Avionics System Example 68
4.1 C&C Views and Specification ALL 68
4.2 C&C Views and Specification ALL $-$ BADS 80
4.3 C&C Views and Specification ALL $-$ HIER 81
4.4 C&C Views and Specification ALL $-$ LAY 84
4.5 C&C Views and Specification ALL $-$ NOLAY 87
4.6 C&C Views and Specification PILOT 88
4.7 C&C Views and Specification PILOT $-$ CS 91

5 Lunar Lander 94
5.1 Lunar Lander Natural Language Description [TMD09] 94
5.2 Lunar Lander Natural Language Description [BS10] 106
Chapter 1

Introduction

MontiArc [HRR12] is an architecture description language (ADL) [MT00] for component and connector (C&C) models. C&C models describe the parallel and hierarchical composition of components and their possible communication via connected interfaces of components.

MontiArcView is an extension of MontiArc with C&C views that can be used to express partial knowledge of an C&C model.

A C&C views specification is a Boolean formula over C&C views.
Chapter 2

Robotic Arm Example

We present an example focusing on the description and specification of the RotationalJoint component of a robotic arm. Each section introduces the new C&C views, the C&C views specification to synthesize, and the result of synthesizing the specification.

2.1 C&C Views and Specification $S_1$

The first specification $S_1$ consists of six C&C views of the robot arm.
The C&C view ASDependence in Listing 2.1 shows the component Body that contains the components Actuator and Sensor. Actuator and Sensor are not necessarily direct subcomponents of Body and could be contained in other subcomponents of Body. Actuator and Sensor are shown side by side on the same component level and thus can not be contained one inside the other.

```java
package robotArmExample;

/**
 * This view shows the actuator and the sensor inside the Body which is
 * considered a bad design since Sensor and Actuator should not be inside
 * a common parent.
 * */

<<view>> component ASDependence {

    component Body {
        component Actuator{}

        component Sensor{}
    }
}
```

Listing 2.1: The C&C view ASDependence showing that component Actuator and component Sensor are both contained in component Body.
The C&C view BodySensorIn in Listing 2.2 shows the component Body and its (not necessarily direct) subcomponents Actuator, Joint, JointLimiter, and Sensor. The incoming ports f1 and f2 of component Actuator (lines 8-9) are marked with the stereotype untyped to indicate that their type is not given in the C&C view. The stereotype needs to be given in order to distinguish untyped from unnamed ports which look similar in concrete syntax.

The C&C view also specifies several abstract connectors (lines 19-23). The connector in line 19 describes that some port of Body sends messages through a chain of connectors to port f1 of component Actuator. Line 21 specifies that a port of component Actuator is connected to a port of component Joint.

```montiarcview
package robotArmExample;

<<view>> component BodySensorIn {

  component Body {
    component Actuator{
      port
      <<untyped>> in f1,
      <<untyped>> in f2;
    }

    component Joint{}

    component JointLimiter{}

    component Sensor{}
  }

  connect Body -> Actuator.f1;
  connect Body -> Actuator.f2;
  connect Actuator -> Joint;
  connect Sensor -> JointLimiter;
  connect JointLimiter -> Actuator;
}
```

Listing 2.2: The C&C view BodySensorIn.
The C&C view BodySensorOut in Listing 2.3 is identical to BodySensorIn in Listing 2.2 except that the component Sensor is now outside the component Body. The abstract connectors are not changed. The connector in line 24 connecting Sensor with JointLimiter now crosses the interface of component Body which contains the JointLimiter.

```java
package robotArmExample;

<<view>> component BodySensorOut {

  component Body {
    component Actuator{
      port
      <<untyped>> in f1,
      <<untyped>> in f2;
    }

    component Joint{}

    component JointLimiter{}
  }

  component Sensor{}

  connect Body -> Actuator.f1;
  connect Body -> Actuator.f2;
  connect Actuator -> Joint;
  connect Sensor -> JointLimiter;
  connect JointLimiter -> Actuator;
}
```

The C&C view Function in Listing 2.4 shows the important components for the function of component RotationalJoint that contains the components Sensor, Actuator and Cylinder. There is a directed connector from a port of component Sensor to a port of component Actuator (line 12). No port is shown on the interface of the components.

```java
package robotArmExample;

<<view>> component Function {
  component RotationalJoint {
    component Sensor{}
    component Actuator{}
    component Cylinder{}
  }
  connect Sensor -> Actuator;
}
```

Listing 2.4: The C&C view Function.
The C&C view SensorConnections in Listing 2.5 shows the components Sensor, Cylinder, and JointLimiter that are in no containment relation. The Sensor has two outgoing ports val1 of type float and val2 of type int (lines 7-8). The first port is connected to a port of component JointLimiter and the second to a port of component Cylinder (lines 15-16).

```
package robotArmExample;

<<view>> component SensorConnections {
  component Sensor{
    port
    out float val1,
    out int val2;
  }

  component Cylinder{}

  component JointLimiter{}

  connect Sensor.val1 -> JointLimiter;
  connect Sensor.val2 -> Cylinder;
}
```

Listing 2.5: The C&C view SensorConnections.
The C&C view Structure in Listing 2.6 shows the main components that are subcomponents of component RotationalJoint. It also shows some connections between the subcomponents.

```java
package robotArmExample;

<<view>> component Structure {

  component RotationalJoint {

    component ServoValve{}

    component Body{}

    component Cylinder{
      port
      in float angle;
    }
  }

  connect RotationalJoint -> ServoValve;
  connect ServoValve -> Body;
  connect Body -> Cylinder.angle;
  connect Cylinder -> Body;
}
```

Listing 2.6: The C&C view Structure.
Specification $S_1$ shown in Table 2.7 uses the six C&C views introduced earlier. It is not desired that the components Sensor and Actuator are both inside component Body. Thus, the view $\text{ASDependence}$ specifying this relation is negated in the specification. The views $\text{BodySensorIn}$ and $\text{BodySensorOut}$ are alternatives since component Sensor should either be inside or outside component Body. All other views are used in as a conjunction meaning a potential C&C model has to satisfy all of them. The number of ports is 18.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$(\neg \text{ASDependence}) \land (\text{BodySensorIn} \lor \text{BodySensorOut}) \land \text{Function} \land \text{SensorConnections} \land \text{Structure}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2.7: Specification $S_1$. 

11
Specification $S_1$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.8.

```plaintext
// Synthesized C&C model for C&C views specification

component RotationalJoint {
  autoinstantiate on;
  port
    out float angle;
}

component Body {
  port
    out float val1,
    in float angle;
}

component Actuator {
  port
    out Type3 val2,
    in float f1,
    in float f2;
}

component Joint {
  port
    in Type3 val1;
}

component JointLimiter {
  port
    in float angle,
    out float f1;
  connect angle -> f1;
  connect JointLimiter.f1 -> val1;
  connect angle -> Actuator.f1;
  connect angle -> Actuator.f2;
  connect angle -> JointLimiter.angle;
  connect Actuator.val2 -> Joint.val1;
}

component Cylinder {
  port
```
in float angle,
out int f2,
in int f1,
out float val2;

connect angle -> val2;
connect f1 -> f2;

} component Sensor {
port
out int val2,
out float val1,
in float f1;

connect f1 -> val1;

} component ServoValve {
port
in float angle,
out float PortName0;

connect angle -> PortName0;

} connect Cylinder.val2 -> angle;
connect Body.val1 -> Sensor.f1;
connect Sensor.val2 -> Cylinder.f1;
connect Sensor.val1 -> Cylinder.angle;
connect ServoValve.PortName0 -> Body.angle;

Listing 2.8: A synthesized C&C model that satisfies specification $S_1$. 
2.2 C&C Views and Specification $S_1 IC$

The C&C view `SensorConnectionsInterfaceComplete` in Listing 2.9 is based on the C&C view `SensorConnections` in Listing 2.5 and adds only the stereotype `interfaceComplete` to component `Sensor`. This states that all ports of component `Sensor` are indeed shown (ports `val1` and `val2`). In a synthesized C&C model `Sensor` may not have additional ports.

```plaintext
package robotArmExample;

<<view>> component SensorConnectionsInterfaceComplete {

  <<interfaceComplete>> component Sensor{
    port
    out float val1,
    out int val2;
  }

  component Cylinder{}

  component JointLimiter{}

  connect Sensor.val1 -> JointLimiter;
  connect Sensor.val2 -> Cylinder;

}
```

Listing 2.9: The C&C view `SensorConnectionsInterfaceComplete`. 

---

[1] MontiArcView

---

14
Specification $S_1IC$ shown in Table 2.10 is identical to specification $S_1$ with C&C view SensorConnections replaced by C&C view SensorConnectionsInterfaceComplete.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$(\neg \text{ASDependence}) \land$ $(\text{BodySensorIn} \lor \text{BodySensorOut}) \land$ Function $\land$ SensorConnectionsInterfaceComplete $\land$ Structure</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2.10: Specification $S_1IC$

Without the stereotype <<interfaceComplete>> the component Sensor in the result of synthesis of specification $S_1$ had four ports (see Listing 2.8, lines 57-61). With the additional restriction in specification $S_1IC$ and C&C view SensorConnectionsInterfaceComplete the component Sensor only has the ports val1 and val2 as specified (see Listing 2.11, lines 59-61).
Specification $S_1 IC$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.11.

```plaintext
// Synthesized C&C model for C&C views specification

class RotationalJoint {  
  autoinstantiate on;  
  port  
    out int f2;
}

class Body {  
  port  
    out float val1,  
    out float f1,  
    in float angle,  
    in int val2;
}

class Actuator {  
  port  
    in float f2,  
    out float PortName0,  
    in float f1;
  
  connect f2 -> PortName0;
}

class Joint {  
  port  
    in float PortName0;
}

class JointLimiter {  
  port  
    out float f2,  
    in float angle;
  
  connect angle -> f2;
}

connect Actuator.PortName0 -> val1;
connect Actuator.PortName0 -> f1;
connect angle -> Actuator.f2;
connect angle -> JointLimiter.angle;
connect Actuator.PortName0 -> Joint.PortName0;
```


connect JointLimiter.f2 -> Actuator.f1;

}  

component Cylinder {
  port
  out int val1,
  in float angle,
  in int f1;

  connect f1 -> val1;

}

component Sensor {
  port
  out int val2,
  out float val1;

}

component ServoValve {
  port
  in int f2,
  out int PortName0;

  connect f2 -> PortName0;

}  

connect Sensor.val2 -> f2;
connect Body.val1 -> Cylinder.angle;
connect Cylinder.val1 -> ServoValve.f2;
connect Sensor.val1 -> Body.angle;
connect ServoValve.PortName0 -> Body.val2;

}  

Listing 2.11: A synthesized C&C model that satisfies specification $S_{1IC}$.  


2.3 C&C Views and Specification $S_1 LC$

Listing 2.12 shows the MontiArc component definition of component ServoValve. A component definition is complete by definition (as opposed to a C&C view). The component ServoValve has the incoming port portIn of type float and the outgoing port portOut of type float. It has no further ports and no subcomponents.

```
package robotArmExample.lib;

component ServoValve {

    port
        in float portIn,
        out float portOut;
}
```

Listing 2.12: The component definition of component ServoValve.
Specification $S_{1LC}$ shown in Table 2.13 is based on specification $S_{1IC}$ but additionally imports component ServoValve from as a library component and uses a larger port scope of 20 instead of 18 ports.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>ServoValve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>($\neg$ ASDependence) $\land$ $(\text{BodySensorIn} \lor \text{BodySensorOut}) \land \text{Function} \land \text{SensorConnectionsInterfaceComplete} \land \text{Structure}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 2.13: Specification $S_{1LC}$
Specification $S_1 LC$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.14.

```蒙提弧
t// Synthesized C&C model for C&C views specification
tcomponent RotationalJoint {
    autoinstantiate on;
    port
        out float val2;
    component Body {
        port
            in float f1,
            out float portIn;
    component Actuator {
        port
            in float f2,
            in float f1,
            out float portIn;
        connect f1 -> portIn;
    } component Joint {
        port
            in float portIn;
    } component JointLimiter {
        port
            out float angle,
            in float portIn;
        connect portIn -> angle;
    } connect f1 -> Actuator.f2;
    connect f1 -> JointLimiter.portIn;
    connect JointLimiter.angle -> portIn;
    connect Actuator.portIn -> Joint.portIn;
    connect JointLimiter.angle -> Actuator.f1;
}
```
component Cylinder {
  port
    in float angle,
    in int val1,
    out float portIn,
    in float portOut,
    in int f2,
    out float f1;
  connect angle -> portIn;
  connect portOut -> f1;
}

component Sensor {
  port
    in float portOut,
    out int val2,
    out float val1;
  component ServoValve {
    port
      out float portOut,
      in float portIn;
  }
  connect portOut -> ServoValve.portIn;
  connect ServoValve.portOut -> val1;
  connect Body.portIn -> val2;
  connect Cylinder.portIn -> Sensor.portOut;
  connect Cylinder.f1 -> Body.f1;
  connect Sensor.val2 -> Cylinder.val1;
  connect Sensor.val2 -> Cylinder.f2;
  connect Sensor.val1 -> Cylinder.portOut;
}

Listing 2.14: A synthesized C&C model that satisfies specification $S_{1}LC$. 
2.4 C&C Views and Specification $S_I^{MP}$

If the component Sensor is placed outside of the component Body as seen in C&C view BodySensorOut in Listing 2.3 the sensor needs an amplifier. View SensorHasAmplifier in Listing 2.15 shows the component Sensor containing a component SensorAmplifier.

The knowledge of the implication can be denoted using the implication pattern with the two views: BodySensorOut → SensorHasAmplifier.

```
package robotArmExample;

<<view>> component SensorHasAmplifier {

    component Sensor {
    
        component SensorAmplifier{}
    }

}
```

Listing 2.15: The C&C view SensorHasAmplifier.
In case component Sensor is inside component Body as shown in C&C view BodySensorIn in Listing 2.2 the sensor needs no amplifier and should not use it.

To prevent the existence of component SensorAmplifier inside a satisfying C&C model we model the C&C view SensorAmplifierView as shown in Listing 2.16. It only shows component SensorAmplifier and its negation will prevent any satisfying C&C model from containing that component.

```
package robotArmExample;

<<view>> component SensorAmplifierView {
  component SensorAmplifier{}
}
```

Listing 2.16: The C&C view SensorAmplifierView.

To express that the inexistence of component SensorAmplifier depends on the C&C view BodySensorIn we use the implication pattern again: BodySensorIn $\rightarrow \neg$ SensorAmplifierView.
Specification $S_1IMP$ as shown in Table 2.17 is based on specification $S1$ and uses the new views and two implications as discussed.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$(\neg \text{ASDependence}) \land$ $(\text{BodySensorIn} \lor \text{BodySensorOut}) \land$ $\text{Function} \land$ $\text{SensorConnections} \land$ $\text{Structure} \land$ $(\text{BodySensorOut} \rightarrow \text{SensorHasAmplifier}) \land$ $(\text{BodySensorIn} \rightarrow \neg \text{SensorAmplifierView})$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2.17: Specification $S_1IMP$. 


Specification $S_1IMP$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.18.

// Synthesized C&C model for C&C views specification
component RotationalJoint {
  autoinstantiate on;
  port
    out float f2;
}

component Body {
  port
    out float f1,
    in float val2,
    in int angle;
}

component Actuator {
  port
    out int PortName0,
    in int f2,
    in float f1;
}

component Joint {
  port
    in int val2;
}

component JointLimiter {
  port
    in float PortName0,
    out float f2;

  connect PortName0 -> f2;
}

connect val2 -> JointLimiter.PortName0;
connect angle -> Actuator.f2;
connect Actuator.PortName0 -> Joint.val2;
connect JointLimiter.f2 -> Actuator.f1;

component Cylinder {
  port
    in float angle,
```java
in int PortName0,
out float f2;
}
component Sensor {
    port
    out int val2,
    out float val1,
    in float angle;
}
component SensorAmplifier {
}
component ServoValve {
    port
    in float f2,
    out int f1;
}
connect ServoValve.f1 -> val2;
connect angle -> ServoValve.f2;
connect angle -> val1;
connect Cylinder.f2 -> f2;
connect Body.f1 -> Cylinder.angle;
connect Sensor.val2 -> Cylinder.PortName0;
connect Sensor.val2 -> Body.angle;
connect Sensor.val1 -> Body.val2;
}
```

Listing 2.18: A synthesized C&C model that satisfies specification $S_{1IMP}$. 
2.5 C&C Views and Specification $S_1 CS$

Specification $S_1 CS$ shown in Table 2.19 is based on specification $S_1$. It specifies that the architecture style for synthesis should be client-server style. We have chosen to identify the server with component Body and the clients with components Cylinder, ServoValve, Actuator, and Sensor.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>Views Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(\neg \text{ASDependence}) \land$</td>
</tr>
<tr>
<td></td>
<td>$(\text{BodySensorIn} \lor \text{BodySensorOut}) \land$</td>
</tr>
<tr>
<td></td>
<td>Function $\land$</td>
</tr>
<tr>
<td></td>
<td>SensorConnections $\land$</td>
</tr>
<tr>
<td></td>
<td>Structure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architectural Style</th>
<th>Client-Server Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Server = Body</td>
</tr>
<tr>
<td></td>
<td>Clients = Cylinder, ServoValve, Actuator, Sensor</td>
</tr>
</tbody>
</table>

| #Ports              | 18                                                                            |

Table 2.19: Specification $S_1 CS$ in client-server style.

Specification $S_1 CS$ is not satisfiable. Synthesis produces no C&C model.

A reason for synthesis to fail to find a satisfying C&C model is that the mandatory C&C view ASDependence requires the component Body contains the components Sensor and Actuator. The client-server style however requires that the server and the clients are disjoint components.
2.6 C&C Views and Specification $S_1\text{HIER}$

Specification $S_1\text{HIER}$ is based on specification $S_1$ and requires that the synthesized C&C model is a hierarchical architecture as defined by Pnueli and Rosner [PR90]. The C&C model is required to contain no directed communication cycles.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>View Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$(\neg \text{ASDependence}) \land$ $\land$ $(\text{BodySensorIn} \lor \text{BodySensorOut}) \land$ $\land$ Function $\land$ $\land$ SensorConnections $\land$ $\land$ Structure</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>Hierarchical Architecture</td>
</tr>
<tr>
<td>#Ports</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2.20: Specification $S_1\text{HIER}$

A directed communication cycle is a cycle that starts at an outgoing port of a component (not forwarded from an input or a subcomponent) and ends at an incoming port of the same component (not forwarded to an output or a subcomponent).
Specification $S_1 HIER$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.21.

```cpp
// Synthesized C&C model for C&C views specification

component RotationalJoint {
    autoinstantiate on;
    port
        in float f1;

component Body {
    port
        in float f1,
        out float val2;

component Actuator {
    port
        out float PortName0,
        in float f1,
        out float val1,
        in float f2;
        connect f2 -> val1;

} component Joint {
    port
        in float val1;

} component JointLimiter {
    port
        in float PortName0,
        out float f2;
        connect PortName0 -> f2;

} connect f1 -> JointLimiter.PortName0;
connect f1 -> Actuator.f2;
connect JointLimiter.f2 -> val2;
connect Actuator.PortName0 -> Joint.val1;
connect JointLimiter.f2 -> Actuator.f1;
```

29
```cpp
} component Cylinder {
  port
    in float angle,
    in int PortName0,
    in float f2,
    out float f1;

  connect f2 -> f1;
}

} component Sensor {
  port
    out int val2,
    out float val1;
}

} component ServoValve {
  port
    out float val2,
    in float PortName0;

  connect PortName0 -> val2;
}

} connect f1 -> ServoValve.PortName0;

} connect Body.val2 -> Cylinder.angle;

} connect Cylinder.f1 -> Body.f1;

} connect Sensor.val2 -> Cylinder.PortName0;

} connect ServoValve.val2 -> Cylinder.f2;
}
```

Listing 2.21: A synthesized C&C model that satisfies specification $S_1^{HIER}$. 

2.7 C&C Views and Specification $S_2$

The C&C view OldDesign in Listing 2.22 shows a partial C&C model where component Body contains the two components Actuator and Cylinder. The component Cylinder has an incoming port angle of type int.

```
package robotArmExample;

<<view>> component OldDesign {
  component Body {
    component Actuator{}
  }
  component Cylinder{
    port
      in int angle;
  }
  connect Actuator -> Cylinder.angle;
}
```

Specification $S_2$ shown in Table 2.23 is based on specification $S_1$ and adds the C&C view $\text{OldDesign}$.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>($\neg \text{ASDependence}) \land$</td>
</tr>
<tr>
<td></td>
<td>($\text{BodySensorIn} \lor \text{BodySensorOut}) \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{Function} \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{SensorConnections} \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{Structure} \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{OldDesign}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2.23: Specification $S_2$

Specification $S_2$ is not satisfiable. Synthesis produces no C&C model.

A reason for the negative result is that C&C view $\text{OldDesign}$ contradicts C&C view $\text{Structure}$ and both need to be satisfied in specification $S_2$. One contradiction is that the C&C view $\text{Structure}$ (see Listing 2.6) states that component $\text{Body}$ and component $\text{Cylinder}$ are independent, while component $\text{Cylinder}$ is shown to be contained in component $\text{Body}$ in C&C view $\text{OldDesign}$. 

32
2.8 C&C Views and Specification $S_2 \text{XCYL}$

The C&C view OldDesignExternalCylinder in Listing 2.24 is based on the C&C view OldDesign but with component Cylinder no longer contained inside component Body.

```java
package robotArmExample;

<<view>> component OldDesignExternalCylinder {

    component Body {
        component Actuator{}
    }

    component Cylinder{
        port
            in int angle;
    }

    connect Actuator -> Cylinder.angle;
}
```

Specification $S_2XCYL$ shown in Table 2.23 is based on specification $S_2$ and replaces the C&C view OldDesign by OldDesignExternalCylinder.

Specification $S_2XCYL$ fixed the inconsistency of the containment between the views Structure and OldDesign regarding the components Body and Cylinder.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
</table>
| Views Formula      | $\neg$ ASDependence $\land$
                  | (BodySensorIn $\lor$ BodySensorOut) $\land$
                  | Function $\land$
                  | SensorConnections $\land$
                  | Structure $\land$
                  | OldDesignExternalCylinder |
| Architectural Style| – |
| #Ports             | 18 |

Table 2.25: Specification $S_2XCYL$

Specification $S_2XCYL$ is not satisfiable. Synthesis produces no C&C model.

A reason for the negative result is that C&C view OldDesignExternalCylinder still contradicts the C&C view Structure since the type of the incoming port angle of component Cylinder is int in C&C view OldDesignExternalCylinder but float in C&C view Structure.
Chapter 3

Pump Station Example

The pump station example shown in this chapter is based on the AutoFOCUS 3.0 pump station example [wwwb].

3.1 C&C Views and Specification *ALL*
package pumpStationExample;

<<view>> component ASPumpingSystem {
    component PumpingSystem {
        component PumpSensorReader {}
        component PumpActuator {
            port in boolean pumpState;
        }
        component ValveSensorReader {}
        component ValveActuator {
            port in ValvePosition valvePosition;
        }
        connect PumpSensorReader -> PumpActuator.pumpState;
        connect ValveSensorReader -> ValveActuator.valvePosition
    }
}

Listing 3.1: The C&C view ASPumpingSystem.
package pumpStationExample;

<<view>> component EnvironmentPhysics {

  component Environment {
    component PhysicsSimulation {
      port
        in boolean valveOpen,
        in boolean valveClose,
        out int level1;
    }

    component SimulationPanel {
      port
        out boolean button;
    }
  }

  component PumpingSystem {
    port
      in int level1,
      in boolean button,
      out boolean valveOpen,
      out boolean valveClose;
  }

  connect PhysicsSimulation.level1 -> PumpingSystem.level1;
  connect SimulationPanel.button -> PumpingSystem.button;
  connect PumpingSystem.valveOpen -> PhysicsSimulation.
    valveOpen;
  connect PumpingSystem.valveClose -> PhysicsSimulation.
    valveClose;
}

package pumpStationExample;

<<view>> component PumpingSystemStructure {
  component PumpingSystem {
    component SensorReading {}
    component PumpActuator {}
    component Controller {}
    component ValveActuator {}
    
    connect SensorReading -> PumpActuator;
    connect SensorReading -> ValveActuator;
    connect SensorReading -> Controller;
    connect Controller -> PumpActuator;
    connect Controller -> ValveActuator;
  }
}

package pumpStationExample;

<<view>> component PumpStationStructure {
    component PumpStation {
        component Environment {}
        component PumpingSystem {}
    }
}

Listing 3.4: The C&C view PumpStationStructure.
package pumpStationExample;
<<view>> component SimulationInput {

  component ValveActuator {
    port
      out boolean valveOpen,
      out boolean valveClose;
  }

  component PumpActuator {} 

  component PhysicsSimulation {
    port
      in boolean valveOpen,
      in boolean valveClose;
  }

  connect ValveActuator.valveOpen -> PhysicsSimulation.
      valveOpen;
  connect ValveActuator.valveClose -> PhysicsSimulation.
      valveClose;
  connect PumpActuator -> PhysicsSimulation;
}

Listing 3.5: The C&C view SimulationInput.
package pumpStationExample;

<<view>> component UserButton {
  component SimulationPanel {
    port
    out boolean button;
  }
  component UserButtonReader {
    port
    in boolean button,
    out UserInput userButton;
  }
  component UserOperation {
    port
    in UserInput userButton,
    out boolean desiredPumpState,
    out ValvePosition desiredValvePosition;
  }
  component ModeArbiter {
    port
    in boolean userPumpState,
    in ValvePosition userValvePosition;
  }
  connect SimulationPanel.button -> UserButtonReader.button;
  connect UserButtonReader.userButton -> UserOperation.
  userButton;
  userPumpState;
  .userValvePosition;
}

Listing 3.6: The C&C view UserButton.
<table>
<thead>
<tr>
<th>Library Components</th>
<th>Views Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASPumpingSystem ∧</td>
</tr>
<tr>
<td></td>
<td>EnvironmentPhysics ∧</td>
</tr>
<tr>
<td></td>
<td>PumpingSystemStructure ∧</td>
</tr>
<tr>
<td></td>
<td>PumpStationStructure ∧</td>
</tr>
<tr>
<td></td>
<td>SimulationInput ∧</td>
</tr>
<tr>
<td></td>
<td>UserButton</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>−</td>
</tr>
<tr>
<td>#Ports</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 3.7: Specification *ALL*
Specification *ALL* is satisfiable and synthesis yields, for example, the C&C model shown in Listing 3.8.

```java
package pumpStationExample;

component PumpStation {
  autoinstantiate on;

  port
  out boolean userButton,
  out boolean valveClose,
  in ValvePosition valveOpen;
}

component Environment {
  port
  in UserInput valveClose,
  in boolean desiredPumpState,
  out int desiredValvePosition,
  in boolean userValvePosition,
  out boolean level1;

  component PhysicsSimulation {
    port
    out int level1,
    in boolean valveClose,
    in boolean valveOpen,
    in UserInput userPumpState;
  }component SimulationPanel {
    port
    out boolean button;
  }

  connect valveClose -> PhysicsSimulation.userPumpState;
  connect desiredPumpState -> PhysicsSimulation.valveOpen;
  connect PhysicsSimulation.level1 -> desiredValvePosition;
  connect userValvePosition -> PhysicsSimulation.valveClose;
  connect SimulationPanel.button -> level1;
}
```
component ModeArbiter {
    port
    in ValvePosition userValvePosition,
    in boolean userPumpState;
}

component PumpingSystem {
    port
    out boolean valveClose,
    out boolean valveOpen,
    in boolean button,
    in int level1,
    out UserInput userPumpState;
}

component Controller {
}

component PumpActuator {
    port
    out UserInput valveOpen;
}

component SensorReading {
    component PumpSensorReader {
    }
}

component ValveActuator {
    port
    out boolean valveClose,
    out boolean valveOpen;
}

component ValveSensorReader {
}
connect ValveActuator.valveOpen -> valveClose;
connect ValveActuator.valveClose -> valveOpen;
connect PumpActuator.valveOpen -> userPumpState;

component UserButtonReader {
    port
    out Type1 userValvePosition,
    out UserInput userButton,
    in UserInput desiredPumpState,
    in boolean button;
    connect desiredPumpState -> userButton;
Listing 3.8: A synthesized C&C model that satisfies specification *ALL.*
3.2 C&C Views and Specification \textit{ALL} – \textit{EMRG}

```java
package pumpStationExample;

// different data type of userPumpState than in other views
// component EmergencyController only mentioned here

<<view>> component SystemEmergencyController {

    component PumpingSystem {

        component EmergencyController {}  
        component UserOperation {}  
        component ModeArbiter {
            port
                in int userPumpState,  
                in ValvePosition valvePosition;
        }

        connect EmergencyController -> UserOperation;
        connect UserOperation -> ModeArbiter.userPumpState;
        connect UserOperation -> ModeArbiter.valvePosition;
    }
}
```


MontiArcView
<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 3.10: Specification $ALL \rightarrow EMRG$

Specification $ALL \rightarrow EMRG$ is not satisfiable. Synthesis produces no C&C model.
3.3 C&C Views and Specification \textit{ALL} – \textit{EMRG} – \textit{FIX}

```java
package pumpStationExample;

<<view>> component SystemEmergencyControllerFixed {

    component PumpingSystem {

        component EmergencyController {}
        component UserOperation {}
        component ModeArbiter {
            port
                in boolean userPumpState,
                in ValvePosition userValvePosition;
        }

        connect EmergencyController -> UserOperation;
        connect UserOperation -> ModeArbiter.userPumpState;
        connect UserOperation -> ModeArbiter.userValvePosition;
    }

```
<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 3.12: Specification $ALL - EMRG - FIX$
Specification \textit{ALL} − \textit{EMRG} − \textit{FIX} is satisfiable and synthesis yields, for example, the C&C model shown in Listing 3.13.

```java
package pumpStationExample;

component PumpStation {
    autoinstantiate on;
    port
        out int valveClose;
}

component Environment {
    port
        out boolean valveClose,
        in boolean valveOpen,
        in int level1,
        in int userPumpState,
        out int button,
        in boolean desiredValvePosition;
}

component PhysicsSimulation {
    port
        in int desiredValvePosition,
        in int desiredPumpState,
        out int level1,
        in boolean valveClose,
        in boolean valveOpen;

    connect desiredValvePosition -> level1;
}

component SimulationPanel {
    port
        out boolean button;

    connect SimulationPanel.button -> valveClose;
    connect valveOpen -> PhysicsSimulation.valveClose;
    connect level1 -> PhysicsSimulation.desiredValvePosition;
    connect userPumpState -> PhysicsSimulation.desiredPumpState;
    connect PhysicsSimulation.level1 -> button;
```

```java
connect desiredValvePosition -> PhysicsSimulation.
   valveOpen;

component PumpingSystem {
   port
      out boolean valveClose,
      out boolean valveOpen,
      in boolean button,
      in int level,
      in int userPumpState,
      out int desiredValvePosition,
      out int userButton;
}

component ModeArbiter {
   port
      in ValvePosition userValvePosition,
      in boolean userPumpState;
}

component Controller {
}

component PumpActuator {
   port
      out int desiredValvePosition;
}

component PumpSensorReader {
   component SensorReading {
   }
}

component UserButtonReader {
   port
      out UserInput userButton,
      in boolean button;
}

component UserOperation {
   port
      out ValvePosition desiredValvePosition,
      out boolean desiredPumpState,
      in UserInput userButton;
}

component ValveActuator {
   port
      out boolean valveClose,
      out boolean valveOpen;
```
component EmergencyController {
}

component ValveSensorReader {
}

connect ValveActuator.valveClose -> valveClose;
connect ValveActuator.valveOpen -> valveOpen;
connect button -> UserButtonReader.button;
connect userPumpState -> desiredValvePosition;
connect PumpActuator.desiredValvePosition -> userButton;
connect UserButtonReader.userButton -> UserOperation.userButton;
connect UserOperation.desiredValvePosition -> ModeArbiter.userValvePosition;
connect UserOperation.desiredPumpState -> ModeArbiter.userPumpState;

}
3.4 C&C Views and Specification PHIS – SIM

```java
package pumpStationExample;

// In this view the component PhysicsSimulation is inside PumpingSystem which
// might not be a good design.
//
//(used as inconsistent view, and negated )

<<view>> component PhysicsInsidePumpingSystem {
  component PumpingSystem {
    component PhysicsSimulation {}
  }
}
```

package pumpStationExample;

// In this view the component PhysicsSimulation is inside PumpingSystem which
// might not be a good design. PhysicsSimulation and Controller should not be
// connected.

<<view>> component PhysicsAndControllerPumpingSystem {
    component PumpingSystem {
        component PhysicsSimulation {}
        component Controller {}
    }

    connect PhysicsSimulation -> Controller;
}

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$\text{EnvironmentPhysics} \land$ $(\text{PhysicsInsidePumpingSystem} \lor \text{PhysicsAndControllerPumpingSystem}) \land \text{SimulationInput}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3.16: Specification $\text{PHIS} – \text{SIM}$

Specification $\text{PHIS} – \text{SIM}$ is not satisfiable. Synthesis produces no C&C model.
### 3.5 C&C Views and Specification $PHIS - SIM2$

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>EnvironmentPhysics ∧ PumpStationStructure ∧ SimulationInput</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3.17: Specification $PHIS - SIM2$
Specification *PHIS − SIM2* is satisfiable and synthesis yields, for example, the C&C model shown in Listing 3.18.

```plaintext
// Synthesized C&C model for C&C views specification

component PumpStation {
  autoinstantiate on;
  port
    out boolean button;
}

component Environment {
  port
    in boolean valveOpen,
    out int button,
    out boolean PortName0,
    in boolean PortName1;
}

component PhysicsSimulation {
  port
    in Type2 button,
    out int level1,
    in boolean valveClose,
    in boolean valveOpen,
    in int PortName0;

  connect PortName0 -> level1;
}

component PumpActuator {
  port
    out boolean PortName1;
}

component SimulationPanel {
  port
    out boolean button;

  connect SimulationPanel.button -> PortName1;
}

connect valveOpen -> PhysicsSimulation.valveClose;
connect PhysicsSimulation.level1 -> button;
connect PumpActuator.PortName1 -> PortName0;
connect PortName1 -> PhysicsSimulation.valveOpen;
```
Listing 3.18: A synthesized C&C model that satisfies specification $PHIS - SIM2$. 
### 3.6 C&C Views and Specification \( PHIS – SIM2 – BADS \)

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>EnvironmentPhysics ( \land ) PumpStationStructure ( \land ) SimulationInput</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.19: Specification \( PHIS – SIM2 – BADS \)

Specification \( PHIS – SIM2 – BADS \) is not satisfiable. Synthesis produces no C&C model.
3.7 C&C Views and Specification \( PHIS - SIM2 - NO - EMRG \)

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>EnvironmentPhysics &amp; PumpStationStructure &amp; SimulationInput &amp; (\neg SystemEmergencyController)</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3.20: Specification \( PHIS - SIM2 - NO - EMRG \)
Specification $PHIS - SIM2 - NO - EMRG$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 3.21.

```plaintext
// Synthesized C&C model for C&C views specification

component PumpStation {
  autoinstantiate on;
  component Environment {
    port
    in boolean userPumpState,
    out boolean valveOpen,
    in boolean button,
    in boolean valveClose,
    out int levell;
  }
  component PhysicsSimulation {
    port
    out int levell,
    in boolean valveClose,
    in boolean valveOpen;
  }
  component PumpActuator {
    port
    in boolean levell,
    out boolean valveClose;
  }
  component SimulationPanel {
    port
    out boolean button,
    in boolean userPumpState;
  }
  component UserOperation {
    }
    connect userPumpState -> button;
  }
  connect levell -> SimulationPanel.userPumpState;
  connect SimulationPanel.button -> valveClose;
}
  connect userPumpState -> PumpActuator.levell;
  connect PumpActuator.valveClose -> valveOpen;
  connect button -> PhysicsSimulation.valveOpen;
```
```java
connect valveClose -> PhysicsSimulation.valveClose;
connect PhysicsSimulation.level1 -> level1;
}
component PumpingSystem {
  port
  out boolean valveClose,
  out boolean valveOpen,
  in boolean button,
  in int level1,
  out boolean userPumpState;
}
component ValveActuator {
  port
    out boolean valveClose,
    out boolean valveOpen,
    in boolean button;
  connect button -> valveClose;
  connect button -> valveOpen;
}
connect ValveActuator.valveOpen -> valveClose;
connect ValveActuator.valveClose -> valveOpen;
connect button -> ValveActuator.button;
connect ValveActuator.valveOpen -> userPumpState;
}
connect Environment.valveOpen -> PumpingSystem.button;
connect Environment.level1 -> PumpingSystem.level1;
connect PumpingSystem.valveClose -> Environment.
valveClose;
connect PumpingSystem.valveOpen -> Environment.
userPumpState;
connect PumpingSystem.userPumpState -> Environment.button
;
}
```

Listing 3.21: A synthesized C&C model that satisfies specification PHIS –
$SIM_2 - NO - EMRG.$
### 3.8 C&C Views and Specification \textit{SENSOR – LIB}

```java
package pumpStationExample.lib;

component UserButtonReader {
    port
        in boolean button,
        out UserInput userButton;
}
```

Listing 3.22: The component definition of component \texttt{UserButtonReader}.

```java
package pumpStationExample.lib;

component ValveSensorReader {
    port
        in int valve,
        out ValvePosition valvePosition;
}
```

Listing 3.23: The component definition of component \texttt{ValveSensorReader}.
<table>
<thead>
<tr>
<th>Library Components</th>
<th>UserButtonReader, ValveSensorReader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>ASPumpingSystem ∧ UserButton ∧ PumpStationStructure ∧ SystemEmergencyControllerFixed</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3.24: Specification $SENSOR - LIB$
Specification $SENSOR - LIB$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 3.25.

```java
// Synthesized C&C model for C&C views specification

class PumpStation {
    autoinstantiate on;
    port
        out boolean valvePosition;
}

class Environment {
}

class PumpingSystem {
    port
        in boolean valvePosition,
        out boolean userButton;
}

class EmergencyController {
    port
        in ValvePosition userValvePosition,
        in boolean userPumpState,
        out boolean userButton;

    connect userPumpState -> userButton;
}

class ModeArbiter {
    port
        in ValvePosition userValvePosition,
        in boolean userPumpState,
        out boolean userButton;

    connect userPumpState -> userButton;
}

class PumpActuator {
    port
        in boolean pumpState,
        out boolean valvePosition,
        out boolean userButton;
}

class SimulationPanel {
    port
        out boolean button,
        in boolean userValvePosition;

    connect userValvePosition -> button;
}

connect pumpState -> SimulationPanel.
    userValvePosition;

connect SimulationPanel.button -> valvePosition;
connect SimulationPanel.button -> userButton;
```
component PumpSensorReader {
}

component UserButtonReader {
  port
    out UserInput userButton,
    in boolean button;
}

component UserOperation {
  port
    in boolean valve,
    out ValvePosition desiredValvePosition,
    out boolean desiredPumpState,
    in UserInput userButton;

  connect valve -> desiredPumpState;
}

component ValveActuator {
  port
    in ValvePosition valvePosition;
}

component ValveSensorReader {
  port
    in int valve,
    out ValvePosition valvePosition;

  connect valvePosition -> PumpActuator.pumpState;
  connect UserOperation.desiredPumpState -> userButton;
  connect ModeArbiter.userButton -> UserOperation.valve;
  connect PumpActuator.valvePosition -> ModeArbiter.
    userPumpState;
  connect UserButtonReader.userButton -> UserOperation.
    userButton;
  connect UserOperation.desiredValvePosition ->
    ModeArbiter.userValvePosition;
  connect UserOperation.desiredPumpState ->
    UserButtonReader.button;
Listing 3.25: A synthesized C&C model that satisfies specification $SENSOR − LIB$. 
Chapter 4

Avionics System Example

The avionics system example presented in this chapter is based on the AADL avionics system example available from the AADL website [wwwa].

4.1 C&C Views and Specification ALL

```
package avionicsSystemExample;

// not consistent with architecture
// all the ports with reverse direction
<<view>> component PilotDisplayManagerPortsReversed {
    // Display_Manager
    component Pilot_DM {
        port
        out Menu_Cmd Menu_selection_from_Display,
        in Page_Image New_Page_Image_To_Display,
        in Page_Request New_Page_Request_To_PCM,
        out Page_Content New_Page_Content_from_PCM;
    }
}
```

package avionicsSystemExample;

// not consistent with architecture

<<view>> component PilotDisplayManagerInsidePilotDisplay {
  // Display
  component Pilot_Display {
    // Display_Manager
    component Pilot_DM {}
  }
}

package avionicsSystemExample;

<<view>> component PilotDisplayManagerIndependentOfPilotDisplay {

    // Display
    component Pilot_Display {} 

    // Display_Manager
    component Pilot_DM {}

}

package avionicsSystemExample;

<<view>> component PilotAndPageContentManager {

    // Display
    component Pilot_Display {
        port
            out Menu_Cmd Menu_Cmd_Pushed;
    }

    // Display_Manager
    component Pilot_DM {
        port
            in Menu_Cmd Menu_selection_from_Display,
            out Page_Request New_Page_Request_To_PCM;
    }

    // Page_Content_Manager
    component PCM {
        port
            in Page_Request New_Page_Request_From_DM;
    }

    connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
    Menu_selection_from_Display;
    connect Pilot_DM.New_Page_Request_To_PCM -> PCM.
    New_Page_Request_From_DM;
}

Listing 4.4: The C&C view PilotAndPageContentManager.
package avionicsSystemExample;

<<view>> component FlightSystemStructure {
    component Flight_System {
        // Display
        component Pilot_Display {
        }
        // Display_Manager
        component Pilot_DM {
        }
        // Page_Content_Manager
        component PCM {
        }
        // Flight_Manager
        component FM {
        }
        // Flight_Director
        component FD {
        }
    }
}

package avionicsSystemExample;

// Shows only the components of the flight control and
their interaction

<<view>> component FlightManagerAndDirector {
  // Flight_Manager
  component FM {
    port
      out Page_Request New_Page_Request_To_FD,
      in Page_Content New_Page_Content_from_FD;
  }

  // Flight_Director
  component FD {
    port
      in Page_Request New_Page_Request_From_FM,
      out Page_Content New_Page_Content_To_FM;
  }

  connect FM.New_Page_Request_To_FD -> FD.
    New_Page_Request_From_FM;
  connect FD.New_Page_Content_To_FM -> FM.
    New_Page_Content_from_FD;
}

Listing 4.6: The C&C view FlightManagerAndDirector.
package avionicsSystemExample;

// Shows that the Pilot_Display exclusively communicates with the Pilot_Display_Manager

<<view>> component DisplayAndManager {
    // Display
    <<interfaceComplete>> component Pilot_Display {
        port
            out Menu_Cmd Menu_Cmd_Pushed,
            in Page_Image Page_To_Show;
    }

    // Display_Manager
    component Pilot_DM {
        port
            in Menu_Cmd Menu_selection_from_Display,
            out Page_Image New_Page_Image_To_Display;
    }

    connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
                     Menu_selection_from_Display;
    connect Pilot_DM.New_Page_Image_To_Display ->
                     Pilot_Display.Page_To_Show;
}

Listing 4.7: The C&C view DisplayAndManager.
package avionicsSystemExample;

// The connection can be used in a positive way (and fail)
// to check that
// interface completeness is enforced
// in a negative way it ...

<<view>> component ConnectPilotDisplayAndPCM {
    // Display
    component Pilot_Display {};

    // Page_Content_Manager
    component PCM {};

    connect PCM -> Pilot_Display;
}

Listing 4.8: The C&C view ConnectPilotDisplayAndPCM.
package avionicsSystemExample;

// inspired by the flow defined for Flight_System.impl

<<view>> component ControlFlowInSystem {
    // Display
    component Pilot_Display {}
    // Display_Manager
    component Pilot_DM {}
    // Page_Content_Manager
    component PCM {}
    // Flight_Manager
    component FM {}
    // Flight_Director
    component FD {}

    connect Pilot_Display -> Pilot_DM;
    connect Pilot_DM -> PCM;
    connect PCM -> FM;
    connect FM -> FD;
    connect FD -> FM;
    connect FM -> PCM;
    connect PCM -> Pilot_DM;
    connect Pilot_DM -> Pilot_Display;
}

Listing 4.9: The C&C view ControlFlowInSystem.
<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$\neg \text{PilotDisplayManagerPortsReversed} \land \neg \text{PilotDisplayManagerInsidePilotDisplay} \land \text{PilotDisplayManagerIndependentOfPilotDisplay} \land \text{PilotAndPageContentManager} \land \text{FlightSystemStructure} \land \text{FlightManagerAndDirector} \land \text{DisplayAndManager} \land \neg \text{ConnectPilotDisplayAndPCM} \land \text{ControlFlowInSystem}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4.10: Specification *ALL*

Specification *ALL* is not satisfiable. Synthesis produces no C&C model.
Specification ALL is satisfiable and synthesis yields, for example, the C&C model shown in Listing 4.11.

```plaintext
// Synthesized C&C model for C&C views specification

component Flight_System {
  autoinstantiate on;
  port
    out Page_Request Page_To_Show;
}

cOMPONENT FD {
  port
    in Page_Request New_Page_Request_From_FM,
    out Page_Content New_Page_Content_To_FM;
}

cOMPONENT FM {
  port
    in Page_Content New_Page_Content_from_FD,
    in Page_Request New_Page_Request_From_DM,
    out Page_Request New_Page_Request_To_FD;
    connect New_Page_Request_From_DM ->
        New_Page_Request_To_FD;
}

cOMPONENT PCM {
  port
    in Page_Request New_Page_Request_From_DM,
    out Page_Request New_Page_Image_To_Display;
    connect New_Page_Request_From_DM ->
       New_Page_Image_To_Display;
}

cOMPONENT Pilot_Display {
  port
    out Menu_Cmd Menu_Cmd_Pushed,
    in Page_Image Page_To_Show;
}

cOMPONENT Pilot_DM {
  port
    in Page_Request New_Page_Content_from_FD,
    in Menu_Cmd Menu_selection_from_Display,
    out Page_Request New_Page_Request_From_DM,
    in Page_Request Page_To_Show,
    out Page_Image New_Page_Image_To_Display,
```
out Page_Request New_Page_Request_To_PCM;

connect New_Page_Content_from_FD -> New_Page_Request_From_DM;

connect Page_To_Show -> New_Page_Request_To_PCM;

connect Pilot_DM.New_Page_Request_To_PCM -> Page_To_Show;

connect FD.New_Page_Content_To_FM -> FM.
       New_Page_Content_from_FD;

connect FM.New_Page_Request_To_FD -> Pilot_DM.
       New_Page_Content_from_FD;

connect PCM.New_Page_Image_To_Display -> FD.
       New_Page_Request_From_FM;

connect PCM.New_Page_Image_To_Display -> Pilot_DM.
         Page_To_Show;

connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
     Menu_selection_from_Display;

connect Pilot_DM.New_Page_Request_From_DM -> PCM.
       New_Page_Request_From_DM;

connect Pilot_DM.New_Page_Image_To_Display ->
     Pilot_Display.Page_To_Show;

Listing 4.11: A synthesized C&C model that satisfies specification \textit{ALL}. 
4.2 C&C Views and Specification $ALL - BADS$

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$\neg \text{PilotDisplayManagerPortsReversed} \land \neg \text{PilotDisplayManagerInsidePilotDisplay} \land \text{PilotDisplayManagerIndependentOfPilotDisplay} \land \text{PilotAndPageContentManager} \land \text{FlightSystemStructure} \land \text{FlightManagerAndDirector} \land \text{DisplayAndManager} \land \neg \text{ConnectPilotDisplayAndPCM} \land \text{ControlFlowInSystem}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 4.12: Specification $ALL - BADS$

Specification $ALL - BADS$ is not satisfiable. Synthesis produces no C&C model.
4.3 C&C Views and Specification $ALL − HIER$

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>($\neg$ PilotDisplayManagerPortsReversed) $\land$ $\neg$ PilotDisplayManagerInsidePilotDisplay $\land$ PilotDisplayManagerIndependentOfPilotDisplay $\land$ PilotAndPageContentManager $\land$ FlightSystemStructure $\land$ FlightManagerAndDirector $\land$ DisplayAndManager $\land$ ($\neg$ ConnectPilotDisplayAndPCM) $\land$ ControlFlowInSystem</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>Hierarchical Architecture</td>
</tr>
<tr>
<td>#Ports</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4.13: Specification $ALL − HIER$
Specification \textit{ALL$-$HIER} is satisfiable and synthesis yields, for example, the C\&C model shown in Listing 4.14.

```montiarc
// Synthesized C\&C model for C\&C views specification

component Flight_System {
    autoinstantiate on;
    port
        out Page_Image New_Page_Content_from_FD;

component FD {
    port
        in Page_Request New_Page_Request_From_FM,
        out Page_Content New_Page_Content_To_FM;
}

component FM {
    port
        in Page_Content New_Page_Content_from_FD,
        in Page_Request Menu_Cmd_Pushed,
        out Page_Request New_Page_Request_To_FD;
    connect Menu_Cmd_Pushed -> New_Page_Request_To_FD;
}

component PCM {
    port
        in Page_Request New_Page_Request_From_DM,
        out Page_Request New_Page_Content_To_FM,
        out Page_Request Page_To_Show;
    connect New_Page_Request_From_DM -> Page_To_Show;
}

component Pilot_Display {
    port
        out Menu_Cmd Menu_Cmd_Pushed,
        in Page_Image Page_To_Show;
}

component Pilot_DM {
    port
        in Page_Request New_Page_Request_To_FD,
        in Menu_Cmd Menu_selection_from_Display,
        in Page_Image Page_To_Show,
        out Page_Image New_Page_Image_To_Display,
        out Page_Request New_Page_Request_To_PCM;
```
connect New_Page_Request_To_FD ->
New_Page_Request_To_PCM;

connect Page_To_Show -> New_Page_Image_To_Display;
}
connect FD.New_Page_Content_To_FM -> FM.
New_Page_Content_from_FD;

connect FM.New_Page_Request_To_FD -> PCM.
New_Page_Request_From_DM;

connect PCM.New_Page_Content_To_FM -> Pilot_DM.
New_Page_Request_To_FD;

connect PCM.Page_To_Show -> FD.New_Page_Request_From_FM;

connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
Menu_selection_from_Display;

connect Pilot_DM.New_Page_Image_To_Display ->
Pilot_Display.Page_To_Show;

connect Pilot_DM.New_Page_Request_To_PCM -> FM.
Menu_Cmd_Pushed;
}

Listing 4.14: A synthesized C&C model that satisfies specification **ALL – HIER**.
4.4 C&C Views and Specification $ALL - LAY$

Specification $ALL - LAY$ shown in Table 4.15 is based on specification $ALL$ but does not include the C&C view FlightSystemStructure. The architectural style desired for the synthesized C&C model is a layered architecture (see, e.g., [TMD09]). Every layer can only communicate with its next previous and next layer. All components need to be contained inside one of the layers. Specification $ALL - LAY$ states that the components Pilot_Display, Pilot_DM, PCM, FM, and FD are identified as layers.

The notation $\text{Pilot\_Display} = C_1, C_2$ could be used to state that layer Pilot_Display contains the components C1 and C2.

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>$(¬ \text{PilotDisplayManagerPortsReversed}) \land$</td>
</tr>
<tr>
<td></td>
<td>$(¬ \text{PilotDisplayManagerInsidePilotDisplay}) \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{PilotDisplayManagerIndependentOfPilotDisplay} \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{PilotAndPageContentManager} \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{FlightManagerAndDirector} \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{DisplayAndManager} \land$</td>
</tr>
<tr>
<td></td>
<td>$(¬ \text{ConnectPilotDisplayAndPCM}) \land$</td>
</tr>
<tr>
<td></td>
<td>$\text{ControlFlowInSystem}$</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>Layered Architecture with layers:</td>
</tr>
<tr>
<td></td>
<td>$\text{Pilot_Display} =$</td>
</tr>
<tr>
<td></td>
<td>$\text{Pilot_DM} =$</td>
</tr>
<tr>
<td></td>
<td>$\text{PCM} =$</td>
</tr>
<tr>
<td></td>
<td>$\text{FM} =$</td>
</tr>
<tr>
<td></td>
<td>$\text{FD} =$</td>
</tr>
<tr>
<td>#Ports</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4.15: Specification $ALL - LAY$
Specification $ALL - LAY$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 4.16.

```montiarc
// Synthesized C&C model for C&C views specification

component LayeredArchitecture {
    autoinstantiate on;
    <<layer>> component Pilot_Display {
        port
            out Menu_Cmd Menu_Cmd_Pushed,
            in Page_Image Page_To_Show;
    }
    <<layer>> component Pilot_DM {
        port
            in Menu_Cmd Menu_selection_from_Display,
            in Page_Request New_Page_Request_To_FD,
            in Page_Request Menu_Cmd_Pushed,
            out Page_Image New_Page_Image_To_Display,
            out Page_Request New_Page_Request_To_PCM;
        connect New_Page_Request_To_FD ->
            New_Page_Request_To_PCM;
    }
    <<layer>> component PCM {
        port
            in Page_Request New_Page_Content_To_FM,
            out Page_Request Page_To_Show,
            in Page_Request New_Page_Request_From_DM,
            out Page_Request Menu_Cmd_Pushed;
        connect New_Page_Content_To_FM -> Menu_Cmd_Pushed;
        connect New_Page_Request_From_DM -> Page_To_Show;
    }
    <<layer>> component FM {
        port
            in Page_Content New_Page_Content_from_FD,
            in Page_Request Page_To_Show,
            out Page_Request New_Page_Request_To_FD;
        connect Page_To_Show -> New_Page_Request_To_FD;
    }
    <<layer>> component FD {
        port
```
in Page_Request New_Page_Request_From_FM,
out Page_Content New_Page_Content_To_FM;
}

connect Pilot_DM.New_Page_Image_To_Display ->
   Pilot_Display.Page_To_Show;
connect Pilot_DM.New_Page_Request_To_PCM -> PCM.
   New_Page_Request_From_DM;
connect PCM.Page_To_Show -> FM.Page_To_Show;
connect PCM.Menu_Cmd_Pushed -> Pilot_DM.Menu_Cmd_Pushed;
connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
   Menu_selection_from_Display;
connect FD.New_Page_Content_To_FM -> FM.
   New_Page_Content_from_FD;
connect FM.New_Page_Request_To_FD -> PCM.
   New_Page_Content_To_FM;
connect FM.New_Page_Request_To_FD -> FD.
   New_Page_Request_From_FM;
}

Listing 4.16: A synthesized C&C model that satisfies specification ALL – LAY.

The C&C model does not start with a single top component as usual but with the component pseudo-component LayeredArchitecture which then contains the layers of the C&C model marked with the stereotype <<layer>>.
### 4.5 C&C Views and Specification \textit{ALL} – \textit{NOLAY}

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightManagerAndDirector ∧ DisplayAndManager ∧ (¬ ConnectPilotDisplayAndPCM) ∧ ControlFlowInSystem ∧ FlightSystemStructure</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>Layered Architecture with layers: Pilot_Display = Pilot_DM = PCM = FM = FD =</td>
</tr>
<tr>
<td>#Ports</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4.17: Specification \textit{ALL} – \textit{NOLAY}

Specification \textit{ALL} – \textit{NOLAY} is not satisfiable. Synthesis produces no C&C model.

The reason for unsatisfiability is that the C&C view FlightSystemStructure from Listing 4.5 specifies that, e.g., component PCM in contained inside component Flight\_System which makes it impossible for component PCM to be a layer (top-level component).
### 4.6 C&C Views and Specification *PILOT*

<table>
<thead>
<tr>
<th>Library Components</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>–</td>
</tr>
<tr>
<td>#Ports</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4.18: Specification *PILOT*
Specification *PILOT* is satisfiable and synthesis yields, for example, the C&C model shown in Listing 4.19.

```montiac
// Synthesized C&C model for C&C views specification

component Flight_System {
  autoinstantiate on;
  port
    out Page_Request New_Page_Request_From_DM,
    in Page_Request New_Page_Image_To_Display,
    in Menu_Cmd Menu_selection_from_Display;

component FD {
}

component FM {
  port
    in Page_Request Menu_Cmd_Pushed;
}

component PCM {
  port
    in Page_Request New_Page_Request_From_DM,
    out Page_Image Menu_Cmd_Pushed,
    out Page_Image New_Page_Content_from_PCM,
    in Page_Request New_Page_Request_To_PCM,
    in Menu_Cmd Menu_selection_from_Display;
}

component Pilot_Display {
  port
    in Page_Request New_Page_Image_To_Display,
    in Page_Request New_Page_Content_from_PCM,
    out Menu_Cmd Menu_Cmd_Pushed,
    out Page_Request Menu_selection_from_Display;

  connect New_Page_Image_To_Display ->
    Menu_selection_from_Display;
}

component Pilot_DM {
  port
    out Page_Request New_Page_Request_To_PCM,
    in Menu_Cmd Menu_selection_from_Display,
    in Page_Request New_Page_Content_from_PCM;
}

connect Pilot_DM.New_Page_Request_To_PCM ->
    New_Page_Request_From_DM;
```
Listing 4.19: A synthesized C&C model that satisfies specification $PILOT$. 
### 4.7 C&C Views and Specification $PILOT - CS$

<table>
<thead>
<tr>
<th>Library Components</th>
<th>PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views Formula</td>
<td>PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure</td>
</tr>
<tr>
<td>Architectural Style</td>
<td>Client-server-architecture with: Server = Pilot_DM Clients = PCM, Pilot_Display</td>
</tr>
<tr>
<td>#Ports</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4.20: Specification $PILOT - CS$
Specification $PILOT - CS$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 4.21.

```montiarc
// Synthesized C&C model for C&C views specification

component ClientServerArchitecture {
  autoinstantiate on;
  <<server>> component Pilot_DM {
    port
      out Menu_Cmd New_Page_Request_From_DM,
      out Page_Request Menu_Cmd_Pushed,
      out Page_Request New_Page_Request_To_PCM,
      in Menu_Cmd Menu_selection_from_Display,
      in Page_Request PortName1,
      in Page_Request PortName0;
    connect Menu_selection_from_Display ->
      New_Page_Request_From_DM;
    connect PortName1 -> Menu_Cmd_Pushed;
    connect PortName1 -> New_Page_Request_To_PCM;
  }
  <<client>> component PCM {
    port
      in Page_Request New_Page_Request_From_DM,
      out Page_Request PortName0,
      out Page_Request New_Page_Request_To_PCM,
      in Page_Request PortName1;
    connect New_Page_Request_From_DM -> PortName0;
  }
  <<client>> component Pilot_Display {
    port
      in Menu_Cmd New_Page_Request_From_DM,
      in Page_Request New_Page_Request_To_PCM,
      out Menu_Cmd Menu_Cmd_Pushed,
      out Menu_Cmd PortName1,
      out Menu_Cmd PortName0,
      out Menu_Cmd Menu_selection_from_Display;
    connect New_Page_Request_From_DM -> Menu_Cmd_Pushed;
  }
  connect Pilot_DM.New_Page_Request_From_DM ->
    Pilot_Display.New_Page_Request_From_DM;
```
connect Pilot_DM.Menu_Cmd_Pushed -> Pilot_Display.
   New_Page_Request_To_PCM;
connect Pilot_DM.New_Page_Request_To_PCM -> PCM.
   New_Page_Request_From_DM;
connect PCM.PortName0 -> Pilot_DM.PortName1;
connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
   Menu_selection_from_Display;
}

Listing 4.21: A synthesized C&C model that satisfies specification PILOT–
CS.
Chapter 5

Lunar Lander

We present C&C views of the structure of a simple lunar lander system. The lunar lander is a space ship with a main controller reading sensor values and employing actuators to safely land the space ship on the surface of the moon.

The lunar lander example is used by Taylor, Medvidovic, and Dashofy throughout their book on software architecture [TMD09]. We present the natural language description of Taylor, Medvidovic, and Dashofy and our derived C&C views in Sect. 5.1. We present a second version of the lunar lander system based on a work by Bagheri and Sullivan [BS10] in Sect. 5.2.

5.1 Lunar Lander Natural Language Description [TMD09]

The basic lunar lander system consists of only three components. The natural language description of the system (taken from [TMD09]) is shown in Figure 5.1.

For this example we have translated the natural language description to eight ArcV views. Each C&C view starts with an excerpt of the natural language description that it represents. The C&C view specification is a conjunct of all views.
**Lunar Lander model in natural language (American English)**

The Lunar Lander application consists of three components: a data store component, a calculation component, and a user interface component.

The job of the data store component is to store and allow other components access to the height, velocity, and fuel of the lander, as well as the current simulator time.

The job of the calculation component is to, upon receipt of a burn rate quantity, retrieve current values of height, velocity, and fuel from the data store component, update them with respect to the input burn rate, and store the new values back. It also retrieves, increments, and stores back the simulator time. It is also responsible for notifying the calling component of whether the simulator has terminated, and with what state (landed safely, crashed, and so on).

The job of the user interface component is to display the current status of the lander using information from both the calculation and the data store components. While the simulator is running, it retrieves the new burn rate value from the user, and invokes the calculation component.

Figure 5.1: Description of the lunar lander system in natural language from [TMD09] (p. 201).
package lunarLanderExample;

// The Lunar Lander application consists of three
// components: a data store component, a
// calculation component, and a user interface
// component.

<<view>> component LunarLanderOverview {
  <<interfaceComplete>> component LunarLanderApplication {
    <<atomic>> component DataStore {}
    <<atomic>> component Calculation {}
    <<atomic>> component UserInterface {}
  }
}

package lunarLanderExample;

// The job of the data store component is to store
// and allow other components access to the height,
// velocity, and fuel of the lander...

<<view>> component DataStoreInfo {
  component DataStore {
    port
      <<untyped>> out height,
      <<untyped>> out velocity,
      <<untyped>> out fuel;
  }
}

Listing 5.3: The C&C view DataStoreInfo.
package lunarLanderExample;

// The job of the data store component is to store
// and allow other components access to ..., as
// well as the current simulator time.

<<view>> component DataStoreInfoSimulation {

cOMPONENT DataStore {
    port
        <<untyped>> out simTime;
    }
}

Listing 5.4: The C&C view DataStoreInfoSimulation.
package lunarLanderExample;

// The job of the calculation component is to, upon
// receipt of a burn rate quantity, retrieve current
// values of height, velocity, and fuel from the
// data store component, update them with respect to
// the input burn rate, and store the new values back.

<<view>> component CalculationJob {

    component DataStore {
        port
        <<untyped>> out height,
        <<untyped>> out velocity,
        <<untyped>> out fuel,
        <<untyped>> in updateHeight,
        <<untyped>> in updateVelocity,
        <<untyped>> in updateFuel;
    }

    component Calculation {
        port
        <<untyped>> in updateBurnRate,
        <<untyped>> in height,
        <<untyped>> in velocity,
        <<untyped>> in fuel,
        <<untyped>> out newHeight,
        <<untyped>> out newVelocity,
        <<untyped>> out newFuel;
    }

    connect DataStore.height -> Calculation.height;
    connect DataStore.velocity -> Calculation.velocity;
    connect DataStore.fuel -> Calculation.fuel;
    connect Calculation.newHeight -> DataStore.updateHeight;
    connect Calculation.newVelocity -> DataStore.updateVelocity;
    connect Calculation.newFuel -> DataStore.updateFuel;
}

Listing 5.5: The C&C view CalculationJob.
package lunarLanderExample;

// It also retrieves, increments, and stores back
// the simulator time.

<<view>> component CalculationJobSimulation {

    component DataStore {
        port
            <<untyped>> in updateSimTime,
            <<untyped>> out simTime;
    }

    component Calculation {
        port
            <<untyped>> in simTime,
            <<untyped>> out newSimTime;
    }

    connect DataStore.simTime -> Calculation.simTime;
    connect Calculation.newSimTime -> DataStore.updateSimTime
    ;
}

Listing 5.6: The C&C view CalculationJobSimulation.
package lunarLanderExample;

// It is also responsible for notifying the calling
// component of whether the simulator has terminated, and
// with what state (landed safely, crashed, and so on).

<<view>> component CalculationJobNotify {

    component Calculation {
        port
            <<untyped>> out callerNotification;
    }

}

Listing 5.7: The C&C view CalculationJobNotify.
package lunarLanderExample;

// The job of the user interface component is to
// display the current status of the lander using
// information from both the calculation and the
// data store components.

<<view>> component UserInterfaceJob {

  component UserInterface { }

  component Calculation { }

  component DataStore { }

  connect Calculation -> UserInterface;
  connect DataStore -> UserInterface;

}

Listing 5.8: The C&C view UserInterfaceJob.
package lunarLanderExample;

// While the simulator is running, it (the user
// interface) retrieves the new burn rate value
// from the user, and invokes the calculation component.

<<view>> component UserInterfaceJobSimulation {

  component UserInterface {
    port
      <<untyped>> out burnRate;
  }

  component Calculation {
    port
      <<untyped>> in updateBurnRate;
  }

  connect UserInterface.burnRate -> Calculation.
    updateBurnRate;
}

Listing 5.9: The C&C view UserInterfaceJobSimulation.
The specification consisting of a conjunction of all views is satisfiable and synthesis yields, for example, the C&C model shown in Listing 5.10 with #Ports set to 22.

```montiarc
// Synthesized C&C model for C&C views specification

component LunarLanderApplication {
    autoinstantiate on;
    component Calculation {
        port
        out Type0 callerNotification,
        in Type1 updateBurnRate,
        in Type1 simTime,
        in Type1 fuel,
        in Type1 velocity,
        in Type1 height,
        out Type0 newSimTime,
        out Type1 newFuel,
        out Type1 newVelocity,
        out Type0 newHeight;
    }
    component DataStore {
        port
        out Type1 fuel,
        out Type1 velocity,
        out Type1 height,
        in Type0 updateSimTime,
        in Type1 updateFuel,
        in Type1 updateVelocity,
        in Type0 updateHeight,
        out Type1 simTime;
    }
    component UserInterface {
        port
        out Type1 height,
        in Type1 updateHeight,
        in Type0 fuel,
        out Type1 burnRate;
    }
}
connect Calculation.newSimTime -> DataStore.updateSimTime;
connect Calculation.newSimTime -> UserInterface.fuel;
```
Listing 5.10: A synthesized C&C model that satisfies the lunar lander specification.
As an additional case of the Lunar Lander we present a second natural language description from Bagheri and Sullivan [BS10] that is based on the previous one from [TMD09]. It further details the sensor and actuator components.

In this application, FlightControl maintains the state of a spacecraft based on the information provided by various sensors: Altimeter, Gyroscope, Fuel level indicator and the engine control switch.

After processing control laws and computing values, FlightControl provides them to the various actuators: Descent engine controller, Attitude control thruster and Display.

Figure 5.11: Description of the lunar lander system in natural language from Bagheri and Sullivan [BS10].
package lunarLanderBS10;

// In this application, FlightControl maintains
// the state of a spacecraft based on the
// information provided by various sensors:
// Altimeter, Gyroscope, Fuel level indicator
// and the engine control switch.

<<view>> component Sensors {
    component LunarLander {
        <<atomic>> component FlightControl{
        <<atomic>> component Altimeter{
        <<atomic>> component Gyroscope{
        <<atomic>> component FuelLevel{
        <<atomic>> component EngineControlSwitch{
        
        connect Altimeter -> FlightControl;
        connect Gyroscope -> FlightControl;
        connect FuelLevel -> FlightControl;
        connect EngineControlSwitch -> FlightControl;
        
    }
}

package lunarLanderBS10;

// After processing control laws and computing values,
// FlightControl provides them to the various actuators:
// Descent engine controller, Attitude control thruster
// and Display.

<<view>> component Actuators {

  component LunarLander {

    <<atomic>> component FlightControl{}

    <<atomic>> component DescentEngineController{}
    <<atomic>> component AttitudeControlThruster{}
    <<atomic>> component Display{}
  }

  connect FlightControl -> DescentEngineController;
  connect FlightControl -> AttitudeControlThruster;
  connect FlightControl -> Display;
}

The C&C views specification consisting of a conjunction of the C&C views Sensors and Actuators is satisfiable and synthesis yields, for example, the C&C model shown in Listing 5.14 with #Ports set to 14.

```plaintext
// Synthesized C&C model for C&C views specification

component LunarLander {
    autoinstantiate on;
    component Altimeter {
        port
        out Type3 PortName4;
    }
    component AttitudeControlThruster {
        port
        in Type3 PortName3;
    }
    component DescentEngineController {
        port
        in Type3 PortName3;
    }
    component Display {
        port
        in Type3 PortName2;
    }
    component EngineControlSwitch {
        port
        out Type3 PortName4;
    }
    component FlightControl {
        port
        in Type3 PortName2,
        out Type3 PortName3,
        in Type3 PortName4,
        in Type3 PortName1,
        in Type3 PortName0;
    }
    component FuelLevel {
        port
        out Type3 PortName3,
        in Type3 PortName4,
        in Type3 PortName2;
    }
}
```
Listing 5.14: A synthesized C&C model that satisfies the lunar lander specification.
Bibliography


