RWTH Aachen University
Software Engineering

MontiArcView Synthesis Evaluation Material
Example Models and Specifications

Shahar Maoz and Jan Oliver Ringert and Bernhard Rumpe

April 17, 2013

Abstract

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

Contents

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.

MontiArcView

```
1  package robotArmExample;
2
3  /**
4    This view shows the actuator and the sensor inside the
         Body which is
5    considered a bad design since Sensor and Actuator should
         not be inside
6    a common parent.
7  **/
8  <<view>> component ASDependence {
9
10     component Body {
11       component Actuator{}
12
13       component Sensor{}
14     }
15
16 }
```

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

```
1  package robotArmExample;
2
3  <<view>> component BodySensorIn {
4
5      component Body {
6        component Actuator{
7          port
8            <<untyped>> in f1,
9            <<untyped>> in f2;
10       }
11
12       component Joint{}
13
14       component JointLimiter{}
15
16       component Sensor{}
17     }
18
19     connect Body -> Actuator.f1;
20     connect Body -> Actuator.f2;
21     connect Actuator -> Joint;
22     connect Sensor -> JointLimiter;
23     connect JointLimiter -> Actuator;
24 }
```

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`.

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component BodySensorOut {
4
5      component Body {
6        component Actuator{
7          port
8            <<untyped>> in f1,
9            <<untyped>> in f2;
10       }
11
12       component Joint{}
13
14       component JointLimiter{}
15
16     }
17
18     component Sensor{}
19
20
21     connect Body -> Actuator.f1;
22     connect Body -> Actuator.f2;
23     connect Actuator -> Joint;
24     connect Sensor -> JointLimiter;
25     connect JointLimiter -> Actuator;
26 }
```

Listing 2.3: The C&C view `BodySensorOut`.

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.

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component Function {
4
5      component RotationalJoint {
6        component Sensor{}
7
8        component Actuator{}
9
10       component Cylinder{}
11     }
12
13     connect Sensor -> Actuator;
14 }
```

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).

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component SensorConnections {
4
5      component Sensor{
6        port
7          out float val1,
8          out int val2;
9      }
10
11     component Cylinder{}
12
13     component JointLimiter{}
14
15     connect Sensor.val1 -> JointLimiter;
16     connect Sensor.val2 -> Cylinder;
17 }
```

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.

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component Structure {
4
5      component RotationalJoint {
6        component ServoValve{}
7
8        component Body{}
9
10       component Cylinder{
11         port
12           in float angle;
13       }
14     }
15
16     connect RotationalJoint -> ServoValve;
17     connect ServoValve -> Body;
18     connect Body -> Cylinder.angle;
19     connect Cylinder -> Body;
20 }
```

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 `ASDependence` specifying this relation is negated in the specification. The views `BodySensorIn` and `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.

Library Components	–
Views Formula	(¬ ASDependence) ∧ (BodySensorIn ∨ BodySensorOut) ∧ Function ∧ SensorConnections ∧ Structure
Architectural Style	–
#Ports	18

Table 2.7: Specification S_1.

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

MontiArc

```
// 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
```

```
46        in float angle,
47        out int f2,
48        in int f1,
49        out float val2;
50
51      connect angle -> val2;
52
53      connect f1 -> f2;
54
55    }
56    component Sensor {
57      port
58        out int val2,
59        out float val1,
60        in float f1;
61
62      connect f1 -> val1;
63
64    }
65    component ServoValve {
66      port
67        in float angle,
68        out float PortName0;
69
70      connect angle -> PortName0;
71
72    }
73    connect Cylinder.val2 -> angle;
74
75    connect Body.val1 -> Sensor.f1;
76
77    connect Sensor.val2 -> Cylinder.f1;
78
79    connect Sensor.val1 -> Cylinder.angle;
80
81    connect ServoValve.PortName0 -> Body.angle;
82
83  }
```

Listing 2.8: A synthesized C&C model that satisfies specification S_1.

2.2 C&C Views and Specification S_1IC

The C&C view `SensorConnectionsInterfaceComplete` in Listing 2.9 is based on the C&C view `SensorConnections` in Listing 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.

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component SensorConnectionsInterfaceComplete {
4
5      <<interfaceComplete>> component Sensor{
6        port
7          out float val1,
8          out int val2;
9      }
10
11     component Cylinder{}
12
13     component JointLimiter{}
14
15     connect Sensor.val1 -> JointLimiter;
16     connect Sensor.val2 -> Cylinder;
17
18 }
```

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

Specification S_1IC shown in Table 2.10 is identical to specification S_1 with C&C view `SensorConnections` replaced by C&C view `SensorConnections-InterfaceComplete`.

Library Components	–
Views Formula	($\neg$ ASDependence) $\wedge$ (BodySensorIn $\vee$ BodySensorOut) $\wedge$ Function $\wedge$ SensorConnectionsInterfaceComplete $\wedge$ Structure
Architectural Style	–
#Ports	18

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_1IC is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.11.

MontiArc

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

component RotationalJoint {
  autoinstantiate on;
  port
    out int f2;

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

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

      connect f2 -> PortName0;

    }
    component Joint {
      port
        in float PortName0;

    }
    component 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;
```

```
46
47      connect JointLimiter.f2 -> Actuator.f1;
48
49    }
50    component Cylinder {
51      port
52        out int val1,
53        in float angle,
54        in int f1;
55
56      connect f1 -> val1;
57
58    }
59    component Sensor {
60      port
61        out int val2,
62        out float val1;
63
64    }
65    component ServoValve {
66      port
67        in int f2,
68        out int PortName0;
69
70      connect f2 -> PortName0;
71
72    }
73    connect Sensor.val2 -> f2;
74
75    connect Body.val1 -> Cylinder.angle;
76
77    connect Cylinder.val1 -> ServoValve.f2;
78
79    connect Sensor.val1 -> Body.angle;
80
81    connect ServoValve.PortName0 -> Body.val2;
82
83  }
```

Listing 2.11: A synthesized C&C model that satisfies specification S_1IC.

2.3 C&C Views and Specification S_1LC

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.

MontiArc

```
1 package robotArmExample.lib;
2
3 component ServoValve {
4
5   port
6     in float portIn,
7     out float portOut;
8 }
```

Listing 2.12: The component definition of component `ServoValve`.

Specification $S1_LC$ 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.

Library Components	ServoValve
Views Formula	$(\neg$ ASDependence$) \wedge$ (BodySensorIn $\vee$ BodySensorOut) $\wedge$ Function $\wedge$ SensorConnectionsInterfaceComplete $\wedge$ Structure
Architectural Style	–
#Ports	20

Table 2.13: Specification S_1LC

Specification S_1LC is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.14.

MontiArc

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

component 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;

  }
```

```
46    component Cylinder {
47      port
48        in float angle,
49        in int val1,
50        out float portIn,
51        in float portOut,
52        in int f2,
53        out float f1;
54
55      connect angle -> portIn;
56
57      connect portOut -> f1;
58
59    }
60    component Sensor {
61      port
62        in float portOut,
63        out int val2,
64        out float val1;
65
66      component ServoValve {
67        port
68          out float portOut,
69          in float portIn;
70
71      }
72      connect portOut -> ServoValve.portIn;
73
74      connect ServoValve.portOut -> val1;
75
76    }
77    connect Body.portIn -> val2;
78
79    connect Cylinder.portIn -> Sensor.portOut;
80
81    connect Cylinder.f1 -> Body.f1;
82
83    connect Sensor.val2 -> Cylinder.val1;
84
85    connect Sensor.val2 -> Cylinder.f2;
86
87    connect Sensor.val1 -> Cylinder.portOut;
88
89  }
```

Listing 2.14: A synthesized C&C model that satisfies specification S_1LC.

2.4 C&C Views and Specification S_1IMP

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` $\rightarrow$ `SensorHasAmplifier`.

MontiArcView

```
1 package robotArmExample;
2
3 <<view>> component SensorHasAmplifier {
4
5   component Sensor {
6     component SensorAmplifier{}
7   }
8
9 }
```

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.

MontiArcView

```
1 package robotArmExample;
2
3 <<view>> component SensorAmplifierView {
4
5     component SensorAmplifier{}
6
7 }
```

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.

Library Components	–
Views Formula	(¬ ASDependence) ∧ (BodySensorIn ∨ BodySensorOut) ∧ Function ∧ SensorConnections ∧ Structure ∧ (BodySensorOut → SensorHasAmplifier) ∧ (BodySensorIn → ¬ SensorAmplifierView)
Architectural Style	–
#Ports	18

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.

MontiArc

```
// 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,
```

```
46        in int PortName0,
47        out float f2;
48
49    }
50    component Sensor {
51      port
52        out int val2,
53        out float val1,
54        in float angle;
55
56      component SensorAmplifier {
57      }
58      component ServoValve {
59        port
60          in float f2,
61          out int f1;
62
63      }
64      connect ServoValve.f1 -> val2;
65
66      connect angle -> ServoValve.f2;
67
68      connect angle -> val1;
69
70    }
71    connect Cylinder.f2 -> f2;
72
73    connect Body.f1 -> Cylinder.angle;
74
75    connect Sensor.val2 -> Cylinder.PortName0;
76
77    connect Sensor.val2 -> Body.angle;
78
79    connect Sensor.val1 -> Body.val2;
80
81  }
```

Listing 2.18: A synthesized C&C model that satisfies specification S_1IMP.

2.5 C&C Views and Specification S_1CS

Specification S_1CS 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`.

Library Components	–
Views Formula	(¬ ASDependence) ∧ (BodySensorIn ∨ BodySensorOut) ∧ Function ∧ SensorConnections ∧ Structure
Architectural Style	Client-Server Architecture Server = Body Clients = Cylinder, ServoValve, Actuator, Sensor
#Ports	18

Table 2.19: Specification S_1CS in client-server style.

Specification S_1CS 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_1HIER

Specification S_1HIER 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.

Library Components	–
Views Formula	(¬ ASDependence) ∧ (BodySensorIn ∨ BodySensorOut) ∧ Function ∧ SensorConnections ∧ Structure
Architectural Style	Hierarchical Architecture
#Ports	18

Table 2.20: Specification S_1HIER

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_1HIER is satisfiable and synthesis yields, for example, the C&C model shown in Listing 2.21.

MontiArc

```
// 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;

```

```
46    }
47    component Cylinder {
48      port
49        in float angle,
50        in int PortName0,
51        in float f2,
52        out float f1;
53
54      connect f2 -> f1;
55
56    }
57    component Sensor {
58      port
59        out int val2,
60        out float val1;
61
62    }
63    component ServoValve {
64      port
65        out float val2,
66        in float PortName0;
67
68      connect PortName0 -> val2;
69
70    }
71    connect f1 -> ServoValve.PortName0;
72
73    connect Body.val2 -> Cylinder.angle;
74
75    connect Cylinder.f1 -> Body.f1;
76
77    connect Sensor.val2 -> Cylinder.PortName0;
78
79    connect ServoValve.val2 -> Cylinder.f2;
80
81  }
```

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`.

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component OldDesign {
4
5      component Body {
6        component Actuator{}
7
8        component Cylinder{
9            port
10           in int angle;
11       }
12     }
13
14     connect Actuator -> Cylinder.angle;
15 }
```

Listing 2.22: The C&C view `OldDesign`.

Specification S_2 shown in Table 2.23 is based on specification S_1 and adds the C&C view `OldDesign`.

Library Components	–
Views Formula	(¬ ASDependence) ∧ (BodySensorIn ∨ BodySensorOut) ∧ Function ∧ SensorConnections ∧ Structure ∧ OldDesign
Architectural Style	–
#Ports	18

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 `OldDesign` contradicts C&C view `Structure` and both need to be satisfied in specification S_2. One contradiction is that the C&C view `Structure` (see Listing 2.6) states that component `Body` and component `Cylinder` are independent, while component `Cylinder` is shown to be contained in component `Body` in C&C view `OldDesign`.

2.8 C&C Views and Specification S_2XCYL

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`.

MontiArcView

```
1  package robotArmExample;
2
3  <<view>> component OldDesignExternalCylinder {
4
5      component Body {
6        component Actuator{}
7      }
8
9      component Cylinder{
10         port
11         in int angle;
12     }
13
14
15     connect Actuator -> Cylinder.angle;
16 }
```

Listing 2.24: The C&C view `OldDesignExternalCylinder`.

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 componennts `Body` and `Cylinder`.

Library Components	–
Views Formula	(¬ ASDependence) ∧ (BodySensorIn ∨ BodySensorOut) ∧ Function ∧ SensorConnections ∧ Structure ∧ 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*

MontiArcView

```
1  package pumpStationExample;
2
3  <<view>> component ASPumpingSystem {
4
5    component PumpingSystem {
6
7     component PumpSensorReader {}
8     component PumpActuator {
9      port
10       in boolean pumpState;
11    }
12    component ValveSensorReader {}
13    component ValveActuator {
14     port
15       in ValvePosition valvePosition;
16    }
17
18    connect PumpSensorReader -> PumpActuator.pumpState;
19    connect ValveSensorReader -> ValveActuator.valvePosition
          ;
20   }
21
22 }
```

Listing 3.1: The C&C view `ASPumpingSystem`.

MontiArcView

```
1  package pumpStationExample;
2
3  <<view>> component EnvironmentPhysics {
4
5    component Environment {
6
7      component PhysicsSimulation {
8        port
9          in boolean valveOpen,
10         in boolean valveClose,
11         out int level1;
12     }
13
14     component SimulationPanel {
15       port
16         out boolean button;
17     }
18   }
19
20   component PumpingSystem {
21     port
22       in int level1,
23       in boolean button,
24       out boolean valveOpen,
25       out boolean valveClose;
26   }
27
28   connect PhysicsSimulation.level1 -> PumpingSystem.level1;
29   connect SimulationPanel.button -> PumpingSystem.button;
30   connect PumpingSystem.valveOpen -> PhysicsSimulation.
       valveOpen;
31   connect PumpingSystem.valveClose -> PhysicsSimulation.
       valveClose;
32 }
```

Listing 3.2: The C&C view `EnvironmentPhysics`.

MontiArcView

```
1  package pumpStationExample;
2
3  <<view>> component PumpingSystemStructure {
4
5    component PumpingSystem {
6
7     component SensorReading {}
8     component PumpActuator {}
9     component Controller {}
10    component ValveActuator {}
11
12    connect SensorReading -> PumpActuator;
13    connect SensorReading -> ValveActuator;
14    connect SensorReading -> Controller;
15    connect Controller -> PumpActuator;
16    connect Controller -> ValveActuator;
17   }
18 }
```

Listing 3.3: The C&C view `PumpingSystemStructure`.

MontiArcView

```
1  package pumpStationExample;
2
3  <<view>> component PumpStationStructure {
4
5    component PumpStation {
6
7     component Environment {}
8     component PumpingSystem {}
9
10   }
11 }
```

Listing 3.4: The C&C view `PumpStationStructure`.

MontiArcView

```
1  package pumpStationExample;
2
3  <<view>> component SimulationInput {
4
5    component ValveActuator {
6      port
7       out boolean valveOpen,
8       out boolean valveClose;
9    }
10
11   component PumpActuator {}
12
13   component PhysicsSimulation {
14     port
15       in boolean valveOpen,
16       in boolean valveClose;
17    }
18
19
20   connect ValveActuator.valveOpen -> PhysicsSimulation.
        valveOpen;
21   connect ValveActuator.valveClose -> PhysicsSimulation.
        valveClose;
22   connect PumpActuator -> PhysicsSimulation;
23
24 }
```

Listing 3.5: The C&C view `SimulationInput`.

MontiArcView

```
1  package pumpStationExample;
2
3  <<view>> component UserButton {
4
5    component SimulationPanel {
6      port
7        out boolean button;
8    }
9
10   component UserButtonReader {
11     port
12       in boolean button,
13       out UserInput userButton;
14   }
15
16   component UserOperation {
17     port
18       in UserInput userButton,
19       out boolean desiredPumpState,
20       out ValvePosition desiredValvePosition;
21   }
22
23   component ModeArbiter {
24     port
25       in boolean userPumpState,
26       in ValvePosition userValvePosition;
27   }
28
29   connect SimulationPanel.button -> UserButtonReader.button
         ;
30   connect UserButtonReader.userButton -> UserOperation.
        userButton;
31   connect UserOperation.desiredPumpState -> ModeArbiter.
        userPumpState;
32   connect UserOperation.desiredValvePosition -> ModeArbiter
        .userValvePosition;
33
34 }
```

Listing 3.6: The C&C view `UserButton`.

Library Components	–
Views Formula	ASPumpingSystem ∧ EnvironmentPhysics ∧ PumpingSystemStructure ∧ PumpStationStructure ∧ SimulationInput ∧ UserButton
Architectural Style	–
#Ports	30

Table 3.7: Specification *ALL*

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

MontiArc

```
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;

  }
```

```
43  component ModeArbiter {
44    port
45      in ValvePosition userValvePosition,
46      in boolean userPumpState;
47
48  }
49  component PumpingSystem {
50    port
51      out boolean valveClose,
52      out boolean valveOpen,
53      in boolean button,
54      in int level1,
55      out UserInput userPumpState;
56
57    component Controller {
58    }
59    component PumpActuator {
60      port
61        out UserInput valveOpen;
62
63    }
64    component SensorReading {
65      component PumpSensorReader {
66      }
67    }
68    component ValveActuator {
69      port
70        out boolean valveClose,
71        out boolean valveOpen;
72
73    }
74    component ValveSensorReader {
75    }
76    connect ValveActuator.valveOpen -> valveClose;
77
78    connect ValveActuator.valveClose -> valveOpen;
79
80    connect PumpActuator.valveOpen -> userPumpState;
81
82  }
83  component UserButtonReader {
84    port
85      out Type1 userValvePosition,
86      out UserInput userButton,
87      in UserInput desiredPumpState,
88      in boolean button;
89
90    connect desiredPumpState -> userButton;
91
```

```
92    }
93    component UserOperation {
94      port
95        out ValvePosition desiredValvePosition,
96        out boolean desiredPumpState,
97        in UserInput userButton;
98
99    }
100   connect PumpingSystem.valveOpen -> userButton;
101
102   connect Environment.level1 -> valveClose;
103
104   connect valveOpen -> ModeArbiter.userValvePosition;
105
106   connect Environment.desiredValvePosition -> PumpingSystem
          .level1;
107
108   connect PumpingSystem.valveClose -> Environment.
          userValvePosition;
109
110   connect PumpingSystem.userPumpState -> Environment.
          valveClose;
111
112   connect UserButtonReader.userButton -> UserOperation.
          userButton;
113
114   connect UserOperation.desiredPumpState -> ModeArbiter.
          userPumpState;
115
116 }
```

Listing 3.8: A synthesized C&C model that satisfies specification *ALL*.

3.2 C&C Views and Specification $ALL - EMRG$

MontiArcView

```
1  package pumpStationExample;
2
3  // different data type of userPumpState than in other views
4  // component EmergencyController only mentioned here
5
6  <<view>> component SystemEmergencyController {
7
8    component PumpingSystem {
9
10    component EmergencyController {}
11    component UserOperation {}
12    component ModeArbiter {
13     port
14       in int userPumpState,
15       in ValvePosition valvePosition;
16    }
17
18    connect EmergencyController -> UserOperation;
19    connect UserOperation -> ModeArbiter.userPumpState;
20    connect UserOperation -> ModeArbiter.valvePosition;
21   }
22 }
```

Listing 3.9: The C&C view `SystemEmergencyController`.

Library Components	–
Views Formula	ASPumpingSystem ∧ EnvironmentPhysics ∧ PumpingSystemStructure ∧ PumpStationStructure ∧ SimulationInput ∧ UserButton ∧ SystemEmergencyController
Architectural Style	–
#Ports	30

Table 3.10: Specification $ALL - EMRG$

Specification $ALL - EMRG$ is not satisfiable. Synthesis produces no C&C model.

3.3 C&C Views and Specification $ALL - EMRG - FIX$

MontiArcView

```
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;
   }
}
```

Listing 3.11: The C&C view `SystemEmergencyControllerFixed`.

Library Components	–
Views Formula	ASPumpingSystem ∧ EnvironmentPhysics ∧ PumpingSystemStructure ∧ PumpStationStructure ∧ SimulationInput ∧ UserButton ∧ SystemEmergencyControllerFixed
Architectural Style	–
#Ports	30

Table 3.12: Specification $ALL - EMRG - FIX$

Specification $ALL - EMRG - FIX$ is satisfiable and synthesis yields, for example, the C&C model shown in Listing 3.13.

MontiArc

```
1  package pumpStationExample;
2
3  component PumpStation {
4    autoinstantiate on;
5
6    port
7      out int valveClose;
8
9    component Environment {
10     port
11       out boolean valveClose,
12       in boolean valveOpen,
13       in int level1,
14       in int userPumpState,
15       out int button,
16       in boolean desiredValvePosition;
17
18     component PhysicsSimulation {
19       port
20         in int desiredValvePosition,
21         in int desiredPumpState,
22         out int level1,
23         in boolean valveClose,
24         in boolean valveOpen;
25
26       connect desiredValvePosition -> level1;
27
28     }
29     component SimulationPanel {
30       port
31         out boolean button;
32
33     }
34     connect SimulationPanel.button -> valveClose;
35
36     connect valveOpen -> PhysicsSimulation.valveClose;
37
38     connect level1 -> PhysicsSimulation.
          desiredValvePosition;
39
40     connect userPumpState -> PhysicsSimulation.
          desiredPumpState;
41
42     connect PhysicsSimulation.level1 -> button;
43
```

```
44      connect desiredValvePosition -> PhysicsSimulation.
            valveOpen;
45
46    }
47    component PumpingSystem {
48      port
49        out boolean valveClose,
50        out boolean valveOpen,
51        in boolean button,
52        in int level1,
53        in int userPumpState,
54        out int desiredValvePosition,
55        out int userButton;
56
57      component ModeArbiter {
58        port
59          in ValvePosition userValvePosition,
60          in boolean userPumpState;
61
62        component Controller {
63        }
64      }
65      component PumpActuator {
66        port
67          out int desiredValvePosition;
68
69      }
70      component PumpSensorReader {
71        component SensorReading {
72        }
73      }
74      component UserButtonReader {
75        port
76          out UserInput userButton,
77          in boolean button;
78
79      }
80      component UserOperation {
81        port
82          out ValvePosition desiredValvePosition,
83          out boolean desiredPumpState,
84          in UserInput userButton;
85
86      }
87      component ValveActuator {
88        port
89          out boolean valveClose,
90          out boolean valveOpen;
91
```

```
92        component EmergencyController {
93        }
94      }
95      component ValveSensorReader {
96      }
97      connect ValveActuator.valveClose -> valveClose;
98
99      connect ValveActuator.valveOpen -> valveOpen;
100
101     connect button -> UserButtonReader.button;
102
103     connect userPumpState -> desiredValvePosition;
104
105     connect PumpActuator.desiredValvePosition -> userButton
            ;
106
107     connect UserButtonReader.userButton -> UserOperation.
            userButton;
108
109     connect UserOperation.desiredValvePosition ->
            ModeArbiter.userValvePosition;
110
111     connect UserOperation.desiredPumpState -> ModeArbiter.
            userPumpState;
112
113   }
114   connect PumpingSystem.desiredValvePosition -> valveClose;
115
116   connect Environment.valveClose -> PumpingSystem.button;
117
118   connect Environment.button -> PumpingSystem.level1;
119
120   connect PumpingSystem.valveClose -> Environment.
        desiredValvePosition;
121
122   connect PumpingSystem.valveOpen -> Environment.valveOpen;
123
124   connect PumpingSystem.desiredValvePosition -> Environment
        .level1;
125
126   connect PumpingSystem.userButton -> Environment.
        userPumpState;
127
128 }
```

Listing 3.13: A synthesized C&C model that satisfies specification $ALL - EMRG - FIX$.

3.4 C&C Views and Specification $PHIS - SIM$

MontiArcView

```
1  package pumpStationExample;
2
3  // In this view the component PhysicsSimulation is inside
       PumpingSystem which
4  // might not be a good design.
5  //
6  //(used as inconsistent view, and negated )
7
8  <<view>> component PhysicsInsidePumpingSystem {
9
10   component PumpingSystem {
11     component PhysicsSimulation {}
12   }
13 }
```

Listing 3.14: The C&C view `PhysicsInsidePumpingSystem`.

MontiArcView

```
1  package pumpStationExample;
2
3  // In this view the component PhysicsSimulation is inside
       PumpingSystem which
4  // might not be a good design. PhysicsSimulation and
       Controller should not be
5  // connected.
6
7  <<view>> component PhysicsAndControllerPumpingSystem {
8
9    component PumpingSystem {
10     component PhysicsSimulation {}
11     component Controller {}
12   }
13
14   connect PhysicsSimulation -> Controller;
15 }
```

Listing 3.15: The C&C view `PhysicsAndControllerPumpingSystem`.

Library Components	–
Views Formula	EnvironmentPhysics ∧ (PhysicsInsidePumpingSystem ∨ PhysicsAndControllerPumpingSystem) ∧ SimulationInput
Architectural Style	–
#Ports	20

Table 3.16: Specification $PHIS - SIM$

Specification $PHIS - SIM$ is not satisfiable. Synthesis produces no C&C model.

3.5 C&C Views and Specification $PHIS - SIM2$

Library Components	–
Views Formula	EnvironmentPhysics ∧ PumpStationStructure ∧ SimulationInput
Architectural Style	–
#Ports	20

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.

MontiArc

```
// 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;

```

```
46    }
47    component ValveActuator {
48      port
49        out boolean valveClose,
50        out boolean valveOpen,
51        in boolean button,
52        in int level1;
53
54      component PumpingSystem {
55        port
56          out boolean valveClose,
57          out boolean valveOpen,
58          in boolean button,
59          in int level1;
60
61        connect button -> valveOpen;
62
63      }
64      connect PumpingSystem.valveClose -> valveClose;
65
66      connect PumpingSystem.valveOpen -> valveOpen;
67
68      connect button -> PumpingSystem.button;
69
70      connect level1 -> PumpingSystem.level1;
71
72    }
73    connect Environment.PortName0 -> button;
74
75    connect Environment.button -> ValveActuator.level1;
76
77    connect Environment.PortName0 -> ValveActuator.button;
78
79    connect ValveActuator.valveClose -> Environment.valveOpen
          ;
80
81    connect ValveActuator.valveOpen -> Environment.PortName1;
82
83  }
```

Listing 3.18: A synthesized C&C model that satisfies specification $PHIS-SIM2$.

3.6 C&C Views and Specification $PHIS - SIM2 - BADS$

Library Components	–
Views Formula	EnvironmentPhysics ∧ PumpStationStructure ∧ SimulationInput
Architectural Style	–
#Ports	10

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 $PHIS-SIM2-NO-EMRG$

Library Components	–
Views Formula	EnvironmentPhysics ∧ PumpStationStructure ∧ SimulationInput ∧ (¬ SystemEmergencyController)
Architectural Style	–
#Ports	20

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.

MontiArc

```
// 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 level1;

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

    }
    component PumpActuator {
      port
        in boolean level1,
        out boolean valveClose;

      component SimulationPanel {
        port
          out boolean button,
          in boolean userPumpState;

        component UserOperation {
        }
        connect userPumpState -> button;

      }
      connect level1 -> SimulationPanel.userPumpState;

      connect SimulationPanel.button -> valveClose;

    }
    connect userPumpState -> PumpActuator.level1;

    connect PumpActuator.valveClose -> valveOpen;

    connect button -> PhysicsSimulation.valveOpen;

```

```
46      connect valveClose -> PhysicsSimulation.valveClose;
47
48      connect PhysicsSimulation.level1 -> level1;
49
50    }
51    component PumpingSystem {
52      port
53        out boolean valveClose,
54        out boolean valveOpen,
55        in boolean button,
56        in int level1,
57        out boolean userPumpState;
58
59      component ValveActuator {
60        port
61          out boolean valveClose,
62          out boolean valveOpen,
63          in boolean button;
64
65        connect button -> valveClose;
66
67        connect button -> valveOpen;
68
69      }
70      connect ValveActuator.valveOpen -> valveClose;
71
72      connect ValveActuator.valveClose -> valveOpen;
73
74      connect button -> ValveActuator.button;
75
76      connect ValveActuator.valveOpen -> userPumpState;
77
78    }
79    connect Environment.valveOpen -> PumpingSystem.button;
80
81    connect Environment.level1 -> PumpingSystem.level1;
82
83    connect PumpingSystem.valveClose -> Environment.
        valveClose;
84
85    connect PumpingSystem.valveOpen -> Environment.
        userPumpState;
86
87    connect PumpingSystem.userPumpState -> Environment.button
        ;
88
89  }
```

Listing 3.21: A synthesized C&C model that satisfies specification $PHIS-$

$SIM2 - NO - EMRG.$

3.8 C&C Views and Specification $SENSOR-LIB$

MontiArc

```
package pumpStationExample.lib;

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

Listing 3.22: The component definition of component `UserButtonReader`.

MontiArc

```
package pumpStationExample.lib;

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

Listing 3.23: The component definition of component `ValveSensorReader`.

Library Components	UserButtonReader, ValveSensorReader
Views Formula	ASPumpingSystem ∧ UserButton ∧ PumpStationStructure ∧ SystemEmergencyControllerFixed
Architectural Style	–
#Ports	20

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.

MontiArc

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

component PumpStation {
  autoinstantiate on;
  port
    out boolean valvePosition;

  component Environment {
  }
  component PumpingSystem {
    port
      in boolean valvePosition,
      out boolean userButton;

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

      connect userPumpState -> userButton;

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

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

        connect userValvePosition -> button;

      }
      connect pumpState -> SimulationPanel.
          userValvePosition;

      connect SimulationPanel.button -> valvePosition;

      connect SimulationPanel.button -> userButton;
```

```
45
46      }
47      component PumpSensorReader {
48      }
49      component UserButtonReader {
50        port
51          out UserInput userButton,
52          in boolean button;
53
54      }
55      component UserOperation {
56        port
57          in boolean valve,
58          out ValvePosition desiredValvePosition,
59          out boolean desiredPumpState,
60          in UserInput userButton;
61
62        connect valve -> desiredPumpState;
63
64      }
65      component ValveActuator {
66        port
67          in ValvePosition valvePosition;
68
69      }
70      component ValveSensorReader {
71        port
72          in int valve,
73          out ValvePosition valvePosition;
74
75      }
76      connect valvePosition -> PumpActuator.pumpState;
77
78      connect UserOperation.desiredPumpState -> userButton;
79
80      connect ModeArbiter.userButton -> UserOperation.valve;
81
82      connect PumpActuator.valvePosition -> ModeArbiter.
            userPumpState;
83
84      connect UserButtonReader.userButton -> UserOperation.
            userButton;
85
86      connect UserOperation.desiredValvePosition ->
            ModeArbiter.userValvePosition;
87
88      connect UserOperation.desiredPumpState ->
            UserButtonReader.button;
89
```

```
90    }
91    connect PumpingSystem.userButton -> valvePosition;
92
93  }
```

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*

MontiArcView

```
1  package avionicsSystemExample;
2
3  // not consistent with architecture
4
5  // all the ports with reverse direction
6
7  <<view>> component PilotDisplayManagerPortsReversed {
8
9    // Display_Manager
10   component Pilot_DM {
11     port
12       out Menu_Cmd Menu_selection_from_Display,
13       in Page_Image New_Page_Image_To_Display,
14       in Page_Request New_Page_Request_To_PCM,
15       out Page_Content New_Page_Content_from_PCM;
16   }
17
18 }
```

Listing 4.1: The C&C view `PilotDisplayManagerPortsReversed`.

MontiArcView

```
1  package avionicsSystemExample;
2
3  // not consistent with architecture
4
5  <<view>> component PilotDisplayManagerInsidePilotDisplay {
6
7    // Display
8    component Pilot_Display {
9      // Display_Manager
10     component Pilot_DM {}
11   }
12 }
```

Listing 4.2: The C&C view `PilotDisplayManagerInsidePilotDisplay`.

MontiArcView

```
1  package avionicsSystemExample;
2
3  <<view>> component
       PilotDisplayManagerIndependentOfPilotDisplay {
4
5    // Display
6    component Pilot_Display {}
7
8    // Display_Manager
9    component Pilot_DM {}
10
11 }
```

Listing 4.3: The C&C view `PilotDisplayManagerIndependentOfPilotDisplay`.

MontiArcView

```
1  package avionicsSystemExample;
2
3  <<view>> component PilotAndPageContentManager {
4
5    // Display
6    component Pilot_Display {
7      port
8        out Menu_Cmd Menu_Cmd_Pushed;
9    }
10
11   // Display_Manager
12   component Pilot_DM {
13     port
14       in Menu_Cmd Menu_selection_from_Display,
15       out Page_Request New_Page_Request_To_PCM;
16   }
17
18   // Page_Content_Manager
19   component PCM {
20     port
21       in Page_Request New_Page_Request_From_DM;
22   }
23
24   connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
       Menu_selection_from_Display;
25   connect Pilot_DM.New_Page_Request_To_PCM -> PCM.
       New_Page_Request_From_DM;
26 }
```

Listing 4.4: The C&C view `PilotAndPageContentManager`.

MontiArcView

```
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 {
    }
  }
}
```

Listing 4.5: The C&C view `FlightSystemStructure`.

MontiArcView

```
1  package avionicsSystemExample;
2
3  // Shows only the components of the flight control and
       their interaction
4
5  <<view>> component FlightManagerAndDirector {
6
7    // Flight_Manager
8    component FM {
9      port
10       out Page_Request New_Page_Request_To_FD,
11       in Page_Content New_Page_Content_from_FD;
12   }
13
14   // Flight_Director
15   component FD {
16     port
17       in Page_Request New_Page_Request_From_FM,
18       out Page_Content New_Page_Content_To_FM;
19   }
20
21   connect FM.New_Page_Request_To_FD -> FD.
         New_Page_Request_From_FM;
22   connect FD.New_Page_Content_To_FM -> FM.
         New_Page_Content_from_FD;
23 }
```

Listing 4.6: The C&C view `FlightManagerAndDirector`.

MontiArcView

```
1  package avionicsSystemExample;
2
3  // Shows that the Pilot_Display exclusively communicates
       with the Pilot_Display_Manager
4
5  <<view>> component DisplayAndManager {
6    // Display
7    <<interfaceComplete>> component Pilot_Display {
8      port
9        out Menu_Cmd Menu_Cmd_Pushed,
10       in Page_Image Page_To_Show;
11   }
12
13   // Display_Manager
14   component Pilot_DM {
15     port
16       in Menu_Cmd Menu_selection_from_Display,
17       out Page_Image New_Page_Image_To_Display;
18   }
19
20   connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
        Menu_selection_from_Display;
21   connect Pilot_DM.New_Page_Image_To_Display ->
        Pilot_Display.Page_To_Show;
22 }
```

Listing 4.7: The C&C view `DisplayAndManager`.

MontiArcView

```
1  package avionicsSystemExample;
2
3  // The connection can be used in a positive way (and fail)
       to check that
4  // interface completeness is enforced
5  // in a negative way it ...
6
7  <<view>> component ConnectPilotDisplayAndPCM {
8
9    // Display
10   component Pilot_Display {}
11
12   // Page_Content_Manager
13   component PCM {}
14
15   connect PCM -> Pilot_Display;
16
17 }
```

Listing 4.8: The C&C view `ConnectPilotDisplayAndPCM`.

MontiArcView

```
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`.

Library Components	–
Views Formula	(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure ∧ FlightManagerAndDirector ∧ DisplayAndManager ∧ (¬ ConnectPilotDisplayAndPCM) ∧ ControlFlowInSystem
Architectural Style	–
#Ports	16

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.

MontiArc

```
// 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,
```

```
44        out Page_Request New_Page_Request_To_PCM;
45
46      connect New_Page_Content_from_FD ->
            New_Page_Request_From_DM;
47
48      connect Page_To_Show -> New_Page_Request_To_PCM;
49
50    }
51    connect Pilot_DM.New_Page_Request_To_PCM -> Page_To_Show;
52
53    connect FD.New_Page_Content_To_FM -> FM.
          New_Page_Content_from_FD;
54
55    connect FM.New_Page_Request_To_FD -> Pilot_DM.
          New_Page_Content_from_FD;
56
57    connect PCM.New_Page_Image_To_Display -> FD.
          New_Page_Request_From_FM;
58
59    connect PCM.New_Page_Image_To_Display -> Pilot_DM.
          Page_To_Show;
60
61    connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
          Menu_selection_from_Display;
62
63    connect Pilot_DM.New_Page_Request_From_DM -> PCM.
          New_Page_Request_From_DM;
64
65    connect Pilot_DM.New_Page_Image_To_Display ->
          Pilot_Display.Page_To_Show;
66
67  }
```

Listing 4.11: A synthesized C&C model that satisfies specification *ALL*.

4.2 C&C Views and Specification $ALL - BADS$

Library Components	–
Views Formula	(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure ∧ FlightManagerAndDirector ∧ DisplayAndManager ∧ (¬ ConnectPilotDisplayAndPCM) ∧ ControlFlowInSystem
Architectural Style	–
#Ports	10

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 $ALL - HIER$

Library Components	–
Views Formula	(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure ∧ FlightManagerAndDirector ∧ DisplayAndManager ∧ (¬ ConnectPilotDisplayAndPCM) ∧ ControlFlowInSystem
Architectural Style	Hierarchical Architecture
#Ports	16

Table 4.13: Specification $ALL - HIER$

Specification $ALL - HIER$ 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;

```

```
46      connect New_Page_Request_To_FD ->
            New_Page_Request_To_PCM;
47
48      connect Page_To_Show -> New_Page_Image_To_Display;
49
50    }
51    connect FD.New_Page_Content_To_FM -> FM.
          New_Page_Content_from_FD;
52
53    connect FM.New_Page_Request_To_FD -> PCM.
          New_Page_Request_From_DM;
54
55    connect PCM.New_Page_Content_To_FM -> Pilot_DM.
          New_Page_Request_To_FD;
56
57    connect PCM.Page_To_Show -> FD.New_Page_Request_From_FM;
58
59    connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
          Menu_selection_from_Display;
60
61    connect Pilot_DM.New_Page_Image_To_Display ->
          Pilot_Display.Page_To_Show;
62
63    connect Pilot_DM.New_Page_Request_To_PCM -> FM.
          Menu_Cmd_Pushed;
64
65  }
```

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 `Pilot_Display = C1, C2` could be used to state that layer `Pilot_Display` contains the components `C1` and `C2`.

Library Components	–
Views Formula	(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightManagerAndDirector ∧ DisplayAndManager ∧ (¬ ConnectPilotDisplayAndPCM) ∧ ControlFlowInSystem
Architectural Style	Layered Architecture with layers: Pilot_Display = Pilot_DM = PCM = FM = FD =
#Ports	16

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
```

```
45          in Page_Request New_Page_Request_From_FM,
46          out Page_Content New_Page_Content_To_FM;
47
48      }
49      connect Pilot_DM.New_Page_Image_To_Display ->
            Pilot_Display.Page_To_Show;
50
51      connect Pilot_DM.New_Page_Request_To_PCM -> PCM.
            New_Page_Request_From_DM;
52
53      connect PCM.Page_To_Show -> FM.Page_To_Show;
54
55      connect PCM.Menu_Cmd_Pushed -> Pilot_DM.Menu_Cmd_Pushed;
56
57      connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
            Menu_selection_from_Display;
58
59      connect FD.New_Page_Content_To_FM -> FM.
            New_Page_Content_from_FD;
60
61      connect FM.New_Page_Request_To_FD -> PCM.
            New_Page_Content_To_FM;
62
63      connect FM.New_Page_Request_To_FD -> FD.
            New_Page_Request_From_FM;
64
65  }
```

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 $ALL - NOLAY$

Library Components	–
Views Formula	(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightManagerAndDirector ∧ DisplayAndManager ∧ (¬ ConnectPilotDisplayAndPCM) ∧ ControlFlowInSystem ∧ FlightSystemStructure
Architectural Style	Layered Architecture with layers: Pilot_Display = Pilot_DM = PCM = FM = FD =
#Ports	16

Table 4.17: Specification $ALL - NOLAY$

Specification $ALL - NOLAY$ 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*

Library Components	–
Views Formula	(¬ PilotDisplayManagerPortsReversed) ∧ (¬ PilotDisplayManagerInsidePilotDisplay) ∧ PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure
Architectural Style	–
#Ports	16

Table 4.18: Specification *PILOT*

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

MontiArc

```
// 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;
```

```
44
45    connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
          Menu_selection_from_Display;
46
47    connect Pilot_Display.Menu_selection_from_Display -> FM.
          Menu_Cmd_Pushed;
48
49    connect Pilot_Display.Menu_selection_from_Display ->
50    PCM.New_Page_Request_From_DM;
51
52    connect Pilot_DM.New_Page_Request_To_PCM ->
53    Pilot_Display.New_Page_Image_To_Display;
54
55 }
```

Listing 4.19: A synthesized C&C model that satisfies specification $PILOT$.

4.7 C&C Views and Specification $PILOT - CS$

Library Components	–
Views Formula	PilotDisplayManagerIndependentOfPilotDisplay ∧ PilotAndPageContentManager ∧ FlightSystemStructure
Architectural Style	Client-server-architecture with: Server = Pilot_DM Clients = PCM, Pilot_Display
#Ports	16

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;
```

```
45
46    connect Pilot_DM.Menu_Cmd_Pushed -> Pilot_Display.
          New_Page_Request_To_PCM;
47
48    connect Pilot_DM.New_Page_Request_To_PCM -> PCM.
          New_Page_Request_From_DM;
49
50    connect PCM.PortName0 -> Pilot_DM.PortName1;
51
52    connect Pilot_Display.Menu_Cmd_Pushed -> Pilot_DM.
          Menu_selection_from_Display;
53
54 }
```

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 savely 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).

MontiArcView

```
1  package lunarLanderExample;
2
3  // The Lunar Lander application consists of three
4  // components: a data store component, a
5  // calculation component, and a user interface
6  // component.
7
8  <<view>> component LunarLanderOverview {
9
10   <<interfaceComplete>> component LunarLanderApplication {
11
12    <<atomic>> component DataStore {}
13
14    <<atomic>> component Calculation {}
15
16    <<atomic>> component UserInterface {}
17  }
18
19 }
```

Listing 5.2: The C&C view `LunarLanderOverview`.

MontiArcView

```
1  package lunarLanderExample;
2
3  // The job of the data store component is to store
4  // and allow other components access to the height,
5  // velocity, and fuel of the lander...
6
7  <<view>> component DataStoreInfo {
8
9    component DataStore {
10     port
11       <<untyped>> out height,
12       <<untyped>> out velocity,
13       <<untyped>> out fuel;
14   }
15 }
```

Listing 5.3: The C&C view `DataStoreInfo`.

MontiArcView

```
1  package lunarLanderExample;
2
3  // The job of the data store component is to store
4  // and allow other components access to ... , as
5  // well as the current simulator time.
6
7  <<view>> component DataStoreInfoSimulation {
8
9    component DataStore {
10     port
11       <<untyped>> out simTime;
12   }
13 }
```

Listing 5.4: The C&C view `DataStoreInfoSimulation`.

MontiArcView

```
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`.

MontiArcView

```
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`.

MontiArcView

```
1  package lunarLanderExample;
2
3  // It is also responsible for notifying the calling
4  // component of whether the simulator has terminated, and
5  // with what state (landed safely, crashed, and so on).
6
7  <<view>> component CalculationJobNotify {
8
9    component Calculation {
10     port
11       <<untyped>> out callerNotification;
12   }
13
14 }
```

Listing 5.7: The C&C view `CalculationJobNotify`.

MontiArcView

```
1  package lunarLanderExample;
2
3  // The job of the user interface component is to
4  // display the current status of the lander using
5  // information from both the calculation and the
6  // data store components.
7
8  <<view>> component UserInterfaceJob {
9
10   component UserInterface { }
11
12   component Calculation { }
13
14   component DataStore { }
15
16   connect Calculation -> UserInterface;
17   connect DataStore -> UserInterface;
18
19 }
```

Listing 5.8: The C&C view `UserInterfaceJob`.

MontiArcView

```
1  package lunarLanderExample;
2
3  // While the simulator is running, it (the user
4  // interface) retrieves the new burn rate value
5  // from the user, and invokes the calculation component.
6
7  <<view>> component UserInterfaceJobSimulation {
8
9    component UserInterface {
10     port
11       <<untyped>> out burnRate;
12   }
13
14   component Calculation {
15     port
16       <<untyped>> in updateBurnRate;
17   }
18
19   connect UserInterface.burnRate -> Calculation.
         updateBurnRate;
20
21 }
```

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;

```

```
43    connect Calculation.newFuel -> DataStore.updateFuel;
44
45    connect Calculation.newVelocity -> DataStore.
          updateVelocity;
46
47    connect Calculation.newHeight -> DataStore.updateHeight;
48
49    connect DataStore.fuel -> Calculation.fuel;
50
51    connect DataStore.velocity -> UserInterface.updateHeight;
52
53    connect DataStore.velocity -> Calculation.velocity;
54
55    connect DataStore.height -> Calculation.height;
56
57    connect DataStore.simTime -> Calculation.simTime;
58
59    connect UserInterface.burnRate -> Calculation.
          updateBurnRate;
60
61 }
```

Listing 5.10: A synthesized C&C model that satisfies the lunar lander specification.

5.2 Lunar Lander Natural Language Description [BS10]

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].

MontiArcView

```
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;

}
```

Listing 5.12: The C&C view `Sensors`.

MontiArcView

```
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;

}
```

Listing 5.13: The C&C view `Actuators`.

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.

MontiArc

```
// 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;
```

```
44
45    }
46    component Gyroscope {
47      port
48        out Type3 PortName2;
49
50    }
51    connect Altimeter.PortName4 -> FlightControl.PortName4;
52
53    connect EngineControlSwitch.PortName4 -> FlightControl.
          PortName1;
54
55    connect FlightControl.PortName3 -> Display.PortName2;
56
57    connect FlightControl.PortName3 ->
          AttitudeControlThruster.PortName3;
58
59    connect FlightControl.PortName3 ->
          DescentEngineController.PortName3;
60
61    connect FuelLevel.PortName3 -> FlightControl.PortName0;
62
63    connect Gyroscope.PortName2 -> FlightControl.PortName2;
64
65    connect Gyroscope.PortName2 -> FuelLevel.PortName4;
66
67  }
```

Listing 5.14: A synthesized C&C model that satisfies the lunar lander specification.

Bibliography

[BS10] Hamid Bagheri and Kevin J. Sullivan. Monarch: Model-based development of software architectures. In *MoDELS (2)*, pages 376–390, 2010.

[HRR12] Arne Haber, Jan Oliver Ringert, and Bernhard Rumpe. MontiArc – Architectural Modeling Of Interactive Distributed and Cyber-Physical Systems. Technical report, RWTH Aachen University, 2012.

[MT00] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 2000.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, pages 746–757, 1990.

[TMD09] Richard N. Taylor, Nenad Medvidovic, and Eric Dashofy. *Software Architecture: Foundations, Theory, and Practice.* Wiley, 2009.

[wwwa] AADL website. `http://www.aadl.info/`. Accessed 2/2012.

[wwwb] AutoFOCUS 3 – The Picture Book. `http://www4.in.tum.de/~ccts/af3/picturebook/af3_picturebook.pdf`.