MontiArcView
Component and Connector Views

Formal Structural Views Specification Language for Component and Connector Models
– Language Overview –

Shahar Maoz
Jan Oliver Ringert
Bernhard Rumpe
September 13, 2013
Abstract

We present a language for component and connector views based on the architecture description language MontiArc for component and connector models.
## Contents

1 Introduction 2

2 C&C models and C&C views 3
   2.1 C&C models .................................................. 3
   2.2 C&C views .................................................... 4
   2.3 Semantics: arc |= view .................................... 6

3 Modeling C&C models using MontiArc 7

4 Modeling C&C Views using MontiArcView 10

Bibliography 13
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 in an architecture and their possible communication via connected interfaces of components.

We present a C&C views specification language that can be used to model partial knowledge of C&C models. C&C views can be applied to modeling abstractions of a given C&C model that only contain model elements of interest for a given feature [GHK+08]. C&C models can also be used in variability modeling for C&C models that are 150% models – every view then can describes a concrete product [GKPR08].

Another use case where it is interesting to be able to express underspecification, e.g., unknown names of ports or types and unknown concrete endpoints of connectors, is the modeling of views as constraints for a C&C model before the C&C model is completely known.

We formally introduce C&C models, C&C views, and a satisfaction relation between C&C models and views in Chapter 2. We give a brief overview of MontiArc’s language elements and concrete syntax in Chapter 3 and present MontiArc’s extension with views in Chapter 4.

We employ an example that is roughly based on the PumpStation example from the AutoFOCUS 3 Picture Book [www12].
Chapter 2

C&C models and C&C views

We give a formal definition of the structures of C&C models and their views with well-formedness rules for valid C&C models and C&C views.

2.1 C&C models

C&C models consist of components at different containment levels, directed, typed, and named ports, and connectors that connect ports. Component and connector models consist of components at different containment levels, directed, typed, and named ports, and connectors that connect ports.

Definition 1 (Component & Connector model). A Component & Connector model \( m \) is a structure \( m = (Cmps, Ports, PNames, Types, Cons, subs, ports, dir, type, pname) \) where

1. \( Cmps \) is a set of named components \( cmp \in Cmps \), each of which has a set of ports \( \text{ports}(cmp) \subseteq Ports \) and a (possibly empty) set of immediate subcomponents \( \text{subs}(cmp) \subset Cmps \),

2. \( Ports \) is a set of directed input and output ports \( p \in Ports \) with \( \text{dir}(p) \in \{IN, OUT\} \) where each port has a name \( \text{pname}(p) \in PNames \), a type \( \text{type}(p) \in Types \), and belongs to exactly one component \( \exists cmp \in Cmps : p \in \text{ports}(cmp) \),

3. \( Cons \) is a set of directed connectors \( con \in Cons \), each of which connects two ports \( \text{con.srcPort} \in \text{ports}(\text{con.srcCmp}) \) and \( \text{con.tgtPort} \in \text{ports}(\text{con.tgtCmp}) \) of the same type \( \text{type}(\text{con.srcPort}) = \text{type}(\text{con.tgtPort}) \), and

4. \( Types \) is a finite set of type names that appear on ports: \( t \in Types \Leftrightarrow \exists p \in Ports : type(p) = t \).

Additionally, the following rules for well-formedness apply:
5. $\exists c \in \text{Cmps} : c \in \text{subs}^+(c)$, where $\text{subs}^+$ denotes the transitive closure of $\text{subs} : \text{Cmps} \times \text{Cmps}$, i.e., no component is its own (transitive) parent,

6. $\forall \text{child} \in \text{Cmps} : |\{\text{parent} \in \text{Cmps} | \text{child} \in \text{subs}($parent$)\}| \leq 1$, i.e., every component has at most one direct parent,

7. $\forall \text{cmp} \in \text{Cmps} : \forall p_1, p_2 \in \text{ports}($cmp$) : \text{pname}(p_1) = \text{pname}(p_2) \Rightarrow p_1 = p_2$, i.e., port names are unique within each component, and

8. $\forall p \in \text{Ports} : |\{\text{con} \in \text{Cons} | \text{con.tgtPort} = p\}| \leq 1$, i.e., every port has at most one incoming connector, and

9. $\forall \text{con} \in \text{Cons}$ we have exactly one of the four cases

(a) $\text{con.srcCmp} = \text{con.tgtCmp} \land \text{dir}($con.srcPort$) \neq \text{dir}($con.tgtPort$)$, i.e., a component forwards input directly as output or feeds back its own output as input,

(b) $\exists \text{parent} \in \text{Cmps} : \text{con.srcCmp} \neq \text{con.tgtCmp} \in \text{subs}($parent$) \land \text{dir}($con.srcPort$) = \text{OUT} \land \text{dir}($con.tgtPort$) = \text{IN}$, i.e., two sibling components are connected,

(c) $\text{con.tgtCmp} \in \text{subs}($con.srcCmp$) \land \text{dir}($con.srcPort$) = \text{IN} \land \text{dir}($con.tgtPort$) = \text{IN}$, i.e., a component forwards input to an immediate child,

(d) $\text{con.srcCmp} \in \text{subs}($con.tgtCmp$) \land \text{dir}($con.srcPort$) = \text{OUT} \land \text{dir}($con.tgtPort$) = \text{OUT}$, i.e., a component forwards output from an immediate child.

Without loss of generality, we consider only C&C models with exactly one top component, i.e., $\exists \text{cmp} \in \text{Cmps} : \exists \text{parent} \in \text{Cmps} : \text{cmp} \in \text{subs}($parent$)$.

Notation: In case the C&C model $m$ is clear from the context, we use the short notation $p \in c.$ports for a port $p$ and component $c$ instead of $p \in m.$ports($c$).

2.2 C&C views

C&C views consist of components at different containment levels, directed, possibly typed, and possibly named ports, and connectors connecting components or ports.

Definition 2 (Component & Connector view). A Component & Connector view is a structure $\text{view} = (\text{Cmps}, \text{Ports}, \text{PNames}, \text{AbsCons}, \text{Types}, \text{subs}, \text{ports}, \text{dir}, \text{type}, \text{pname}, \text{stereotypes})$ where
1. **Cmps** is a set of named components \( cmp \in \text{Cmps} \), each of which has a set of ports \( \text{ports}(cmp) \subseteq \text{Ports} \) and a set of subcomponents \( \text{subs}(cmp) \subset \text{Cmps} \).

2. **Ports** is a set of directed input and output ports \( p \in \text{Ports} \) with \( \text{dir}(p) \in \{\text{IN}, \text{OUT}\} \) where each port has a (possibly unknown) name \( \text{pname}(p) \in \text{PNames} \cup \{\bot\} \), a (possibly unknown) type \( \text{type}(p) \in \text{Types} \cup \{\bot\} \), and belongs to exactly one component \( \exists cmp \in \text{Cmps} : p \in \text{ports}(cmp) \).

3. **AbsCons** is a set of directed abstract connectors \( \text{con} \in \text{Cons} \), each of which connects two components optionally via their ports (\( \text{con.srcPort} \in \text{ports}(\text{con.srcCmp}) \lor \text{con.srcPort} = \bot \) and \( \text{con.tgtPort} \in \text{ports}(\text{con.tgtCmp}) \lor \text{con.tgtPort} = \bot \)), connected ports have the same or an unknown type (\( |\{\text{type}(\text{con.srcPort}), \text{type}(\text{con.tgtPort})\}\setminus\{\bot\}| \leq 1 \)), and

4. **Types** is a finite set of type names that appear on ports: \( t \in \text{Types} \Leftrightarrow \exists p \in \text{Ports} : \text{type}(p) = t \).

Additionally, the following rules for well-formedness apply:

5. \( \nexists c \in \text{Cmps} : c \in \text{subs}^+(c) \), where \( \text{subs}^+ \) denotes the transitive closure of \( \text{subs} : \text{Cmps} \times \text{Cmps} \), i.e., no component is its own (transitive) parent,

6. \( \forall \text{child} \in \text{Cmps} : |\{\text{parent} \in \text{Cmps} \mid \text{child} \in \text{subs}($\text{parent}$)$\}| \leq 1 \), i.e., every component has at most one direct parent in the view, and

7. \( \forall \text{cmp} \in \text{Cmps} : \forall p_1, p_2 \in \text{ports}($\text{cmp}$) : \bot \neq \text{pname}(p_1) = \text{pname}(p_2) \Rightarrow p_1 = p_2 \), i.e., known port names are unique within each component.

Note that in a C&C view, abstract connectors are not restricted to the four cases of connectors in C&C models as defined in Definition 1, Item 9. Rather, they may connect between any two ports and components in the view. Also, ports may have multiple incoming connectors in C&C views since these might refer to the same concrete chain of connectors in a C&C model. We do not restrict C&C views to have exactly one top component.

**Notation:** In case the C&C view \( \text{view} \) is clear from the context, we use the short notation \( p \in \text{c.ports} \) for a port \( p \in \text{Ports} \) and component \( c \in \text{Cmps} \) instead of \( p \in \text{view.ports}(c) \). In addition, we write \( \text{view.name} \) and \( \text{c.name} \) (for \( c \in \text{Cmps} \)) for the unique names of views and components.

C&C models and C&C views share the same structural elements, i.e., hierarchically composed components and directed, named, and typed ports. They however differ in the amount of required information specified about these elements and in their semantics.

\[\footnote{We use \( \bot \) to denote unknown names, types, and ports.}\]
2.3 Semantics: arc $\models$ view

We are now ready to define the semantics of C&C views, specifically, when does a C&C model satisfy a C&C view.

Roughly, a C&C model satisfies a C&C view iff the types, components, and ports mentioned in the C&C view are contained in the C&C model, the C&C model respects the subcomponent relation induced by the C&C view, two ports connected by an abstract connector in the C&C view are connected by a chain of connectors in the C&C model (respecting direction, names, and types), and all ports of a component in the C&C view belong to the same component in the C&C model with corresponding name, type and direction. More formally:

**Definition 3 (C&C model $\models$ C&C view).** A C&C model $m$ satisfies a C&C view $\text{view}$ iff:

1. $\text{view.Types} \subseteq m.\text{Types}$,
2. $\text{view.Cmps} \subseteq m.\text{Cmps}$,
3. $\forall \text{cmp}_1, \text{cmp}_2 \in \text{view.Cmps}$: $\text{cmp}_1 \in \text{view.subs}(\text{cmp}_2)$ iff $\text{cmp}_1 \in m.\text{subs}^+(\text{cmp}_2)$ (we use $^+$ to denote the transitive closure),
4. $\forall \text{cmp} \in \text{view.Cmps}$:
   (a) $\forall p \in \text{view.ports}(\text{cmp})$ $\exists p' \in \text{m.ports}(\text{cmp}) : p \equiv p'$ where
   (b1) $\text{view.dir}(p) = m.\text{dir}(p')$ ∧
   (b2) $\text{view.type}(p) \in \{\bot, m.\text{type}(p')\}$ ∧
   (b3) $\text{view.pname}(p) \in \{\bot, m.\text{pname}(p')\}$, and
5. $\forall \text{ac} \in \text{view.AbsCons}$ $\exists c_1, \ldots, c_n \in m.\text{Cons}$ such that
   (a) $\text{ac.srcCmp} = c_1.\text{srcCmp} \land (\text{ac.srcPort} \cong c_1.\text{srcPort} \lor \text{ac.srcPort} = \bot) \land$
   (b) $\text{ac.tgtCmp} = c_n.\text{tgtCmp} \land (\text{ac.tgtPort} \cong c_n.\text{tgtPort} \lor \text{ac.tgtPort} = \bot) \land$
   (c) $\forall 1 \leq i < n : c_i.\text{tgtPort} = c_{i+1}.\text{srcPort}$.

Note that by definition the empty view is satisfied by any C&C model.
Chapter 3

Modeling C&C models using MontiArc

MontiArc is an architecture description language (ADL) developed using the DSL framework MontiCore [KRV10]. For more information on MontiArc see [HRR12, www11]. We only give a brief overview of MontiArc for modeling C&C models. The following description is based on the MontiArc language reference [HRR12] with additional examples of MontiArc’s concrete syntax.

Basic MontiArc Elements

The basic elements of MontiArc are components with ports, subcomponents, and connectors that unidirectional connect ports of components. The interface of a component is a set of typed and directed ports. The internal structure of components can be defined by decomposition to subcomponents and their composition via connector definitions.

In this report we ignore the language features component types, referencing and instantiation, generic component types, and configurable components that are described in the MontiArc technical report [HRR12]. Without using these features the example shown is still a valid MontiArc component definition that can be used, e.g., for code generation.

Components and Subcomponents

Listing 3.1 shows the C&C model of component PumpStation (line 3) that contains the components Environment (line 5) and UserButtonReader (line 22) as direct sub components. The subcomponent Environment consists of the components PhysicsSimulation and SimulationPanel. The subcomponent UserButtonReader of component PumpStation is no further decomposed.
Component definitions are introduced using the keyword `component`. In this simplified case there is no syntactical difference between subcomponent and component definitions. Subcomponents are defined inside the body of their parent component (see, e.g., lines 9 and 13 of Listing 3.1).

```montiarc
package pumpStationExample;

component PumpStation {
    autoInstantiate on;

    component Environment {
        port
            out boolean button;

    component PhysicsSimulation {
        // ...
    }

    component SimulationPanel {
        port
            in int level1,
            // ...
            out Button button;
        }
        connect SimulationPanel.button -> button;

    connect Environment.button -> UserButtonReader.button;

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

Listing 3.1: A model of the component PumpStation given in MontiArc syntax.
```

**Interfaces and Ports**

The interface of component `Environment` (lines 5-7) consists of the single outgoing port with name `button` which sends messages of type `boolean`. Multiple definitions of ports can be done each starting with the keyword
port inside the body of the component or in a comma separated list as shown in lines 26 and 27.

Connectors

The connector in line 19 of Listing 3.1 connects the port button of the subcomponent SimulationPanel with the port button of the parent component Environment. When defining C&C models, connectors always have to be placed inside their parent component and use the ports of the parent unqualified while qualifying subcomponent’s ports with their component name.

For the purpose of presentation we only consider C&C models that are – with all subcomponents and connectors – completely defined within one artifact. Please note that MontiArc also supports the definition in separate files down to one component per file and suitable import mechanisms that are essential for the reuse of models.
Chapter 4

Modeling C&C Views using MontiArcView

The language MontiArcView for C&C views extends the MontiArc language with several stereotypes.

Defining a View

Every view definition has to be marked with the stereotype \texttt{<<view>>} which can only be applied to the top level component definition. While in C&C models the first component definition is the root of the system the stereotype \texttt{<<view>>} around the outer definition does not resemble a component but a view which is a collection of the components of interest and the relations known between them. Line 3 in Listing 4.1 defines the C&C view UserButton.

Containment, Independence, and Atomic Components

A C&C views the relation of components with each other and their properties in satisfying C&C models.

Independence of Components

The C&C view given in Listing 4.1 shows the view UserButton which contains the two components SimulationPanel and UserButtonReader. In a C&C view the knowledge about the C&C model is partial. For example the component UserButtonReader does not necessarily need to be a sibling component of component SimulationPanel. The two components are however shown side by side meaning that they are not contained in each other in a concrete C&C model.

Component Containment

The component Sensor (line 10) is shown to be contained in component UserButtonReader. This containment is not necessarily concrete – component Sensor might be a direct subcom-
Listing 4.1: A C&C view of the system called UserButton

ponent of UserButtonReader or contained in any of its subcomponents that might exist but are not shown inside the C&C view.

Atomic Components The stereotype <<atomic>> in front of component Sensor (line 10) marks the component to be atomic. This stereotype documents additional knowledge of the engineer: the component may not have subcomponents or internal connectors in any satisfying C&C model. The atomicity property does not state anything about the interface of the component.

Component Interface

The interfaces of components can be underspecified in C&C views. Interfaces in a component definition can be either completely omitted, contain untyped or unnamed ports, or can be marked as complete.

Missing Interface If a component in a C&C view has no definition of an interface (starting with keyword port) nothing is known about the component’s interface. The interface is neither restricted nor required to contain any ports.

Untyped Port The stereotype <<untyped>> in front of the port of a component, e.g., the first port of component UserButtonReader in Listing 4.1 indicates that the type of the port (here: button, line 7) is not known in the view. In a satisfying C&C model there will be an incoming port with name button and a concrete type.
**Unnamed Port**  The stereotype `<<unnamed>>` on the second port of component `UserButtonReader` means that the name of the port is not known in the view. Its direction (outgoing) and type (`UserInput`) are however known.

**Interface Complete**  The stereotype `<<interfaceComplete>>` in front of the component `SimulationPanel` means that its interface is completely specified in the model and thus consists of the single outgoing port `button` of type `Button`. When using the stereotype `<<interfaceComplete>>` all ports in the interface of the component have to be listed at least with a name and direction. In this case it is possible to use the stereotype `<<untyped>>` but not the stereotype `<<unnamed>>` for individual ports.

In all cases (unnamed or untyped) the direction of the port has to be given inside the views.

```java
package pumpStationExample;

<<view>> component UserButtonWithConnections {

  component UserButtonReader {
    port
      in Button button;
  }

  component SimulationPanel {
    port
      out Button button;

    component PreProcessor {
      port
        <<untyped>> out trans;
    }
  }

  component Environment {
  }

  connect SimulationPanel -> Environment;
  connect UserButtonReader -> SimulationPanel.button;
  connect PreProcessor.trans -> Environment;
}
```

Listing 4.2: A C&C view of the system called `UserButtonWithConnections`
Abstract Connectors

In C&C views connections are not defined on the level of the containing component but the level of the view. Listing 4.2 shows the definition of abstract connectors.

Component-to-Component  Line 23 specifies that component SimulationPanel and component Environment are connected. In C&C models components can only be connected via corresponding ports – these are not required to be explicitly given in C&C views. Required ports, e.g., an incoming port of component Environment can be omitted in C&C views as long as their name is not used in the definition of a connector.

Component-to-Port  Line 24 specifies a connection between component UserButtonReader and component SimulationPanel. It is known that the endpoint of the connector is port button of component SimulationPanel. The same connector form is also possible for Port-to-Component connectors (line 25).

Component Crossing Connectors  Line 25 specifies a connection between component PreProcessor’s port trans and component Environment. In this C&C view the connector crosses the border of component SimulationPanel that contains component Environment. Similarly, if the components UserButtonReader and SimulationPanel are not siblings in a concrete C&C model the connector in line 24 might cross other component’s interfaces requiring corresponding ports and further connectors not shown in the C&C view.

Abstract connectors in C&C views specify a chain of connectors in the C&C model – the data is forwarded as is: no intermediate processing by components is allowed unless these only forward the data by suitable connectors.
Bibliography


