FI MU

Faculty of Informatics
Masaryk University Brno

VCD: A Visual Formalism for Specification of Heterogeneous Software Architectures

by

David Šafránek
Jiří Šimša

FI MU Report Series **FIMU-RS-2004-11**

 December 2004

Publications in the FI MU Report Series are in general accessible via WWW:

`http://www.fi.muni.cz/veda/reports/`

Further information can obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**

VCD: A Visual Formalism for Specification of Heterogeneous Software Architectures*†

David Šafránek and Jiří Šimša
Faculty of Informatics, Masaryk University Brno
Czech Republic
{xsafran1,xsimsa}@fi.muni.cz

Abstract

A visual formalism called Visual Coordination Diagrams (VCD) for high-level design of heterogeneous systems is presented in this paper. The language is based on a state-transition operational semantics, which allows application of formal methods to software design. Formal definition of VCD is included in the paper. Moreover, an example of use of the language is also given.

1 Introduction

The importance of visual modeling languages such as UML [14] is very significant in the domain of software engineering. The desired properties of such an universal visual design language are heterogeneity, hierarchy and component-based structure. Additionally, to be able to analyze the software design using formal methods, some unambiguous formal semantics is required. Unfortunately, there is no formal semantics of UML [8].

In this paper we present Visual Coordination Diagrams (VCD) – a *visual formalism* for specification of component-based distributed systems, based on the idea of GCCS [4] and its extensions [20]. The VCD formalism can be viewed as static architecture diagrams for specification of connections among components. The key property of VCD is its two-level *heterogeneity*. The first level of this heterogeneity is based on the possibility

*This work has been supported by the Grant Agency of Czech Republic grant No. 201/03/0509.
†This is a full version of SOFSEM 2005 paper.

of combination of various coordination models (both synchronous and asynchronous) in a particular specification. The second level of the heterogeneity is the variability of specification of behavioral aspects. This can be done in various notations which have to be, in some well-defined sense, compatible with the supported coordination models.

The work on VCD is practically motivated by the formal verification project Liberouter [2]. In this project, we have to deal with formal modeling of a complex system composed of heterogeneous SW/HW units [9].

1.1 Background and Related Work

There is a group of visual languages for concurrent systems in which the classical state transition diagrams have been extended to fulfill the needs of design of complex systems. Combining the concept of geometric inclusion with the concept of hi-graphs, the hierarchy of states has been added, leading to Harel's Statecharts [6]. The complexity of the syntactic richness of Statecharts has shown that reaching a compositional formal semantics for such a powerful language is impracticable. Various sub-dialects of Statecharts have been defined to achieve required semantic properties [12]. The concept of Statecharts was also incorporated in UML [19], [7].

Another group of visual languages is based on the concept of message flow graphs. They are employed to visually describe partial message passing interaction among concurrent processes. The high level message flow diagrams called Message Sequence Charts are based on this concept [10]. This notation does not support hierarchical design. For its simple nature, it is widely used in telecommunication industry and it is also a part of UML.

VCD extends and generalizes ideas of the work on Graphical calculus of communicating systems (GCCS) [4] and its synchronous extension SGCCS [20], which adopt the process algebraic approach as the underlying semantic model. In these languages, a very tight relation to the underlying process algebraic semantic model limits the heterogeneity of both coordination and behavioral layers. I.e., it is difficult to incorporate Statechart-like formalisms into GCCS. In VCD we try to overcome these inconveniences by using of a more general semantic model.

There is another architectural language, which is, similarly to VCD, based on the idea of GCCS. It is called Architectural Interaction Diagrams (AID) [17]. VCD and AID both achieve some level of heterogeneity by avoiding the tight relation with the CCS process algebra [13]. One of the significant differences between these two formalisms

is in the underlying semantic model. AID is aimed to be used for specification of interactive systems while in VCD the interactive aspects can be additionally mixed with reactivity. At the behavioral layer, VCD supports more expressive formalisms than AID, and thus allows more heterogeneity at this level.

Similarly to some of the classical textual architecture description languages (ADLs) like Wright [1] or UniCON [18], VCD are based on the idea of taking connectors and computational components as different elements of system architecture. Moreover, this concept is further refined in VCD. In contrast to Wright, where the semantics of connectors is defined in terms of CSP processes, which are based on handshake-style coordination, in VCD more complex coordination mechanisms, e.g. multi-cast, can be modeled more conveniently. It is mainly due to fact of semantic modeling of any coordination event, e.g. a broad-cast communication, as one atomic transition of a connector model. In the domain of ADLs, there are some languages which support dynamic changes of system structure, i.e. Darwin [11] or SOFA [16]. Unlike these languages, VCD does not support dynamism. VCD is aimed to be a simple visual formalism for hierarchical description of coordination of system components.

The main reason for developing VCD is our belief in importance of building a formal framework for coordination of various kinds of Statecharts and other visual formalisms for specification of component behavior. We would like to establish a simple syntactic visual notation with suitable underlying formal semantics. The chosen semantic model is based on composition of local transition systems, which represent particular components, resulting in one global transition system formally representing the whole architecture description.

2 Overview of VCD

VCD is aimed to be a formal language for specification of communication relationships in component-based systems. An example of a simple VCD is depicted in Fig. 1. Basic elements of the VCD formalism are component *interfaces*. Each interface contains input and output *ports*. Interfaces are organized in so called *networks* in which they can be connected by *links* to *buses*.

Buses represent connectors of components. They are used for specification of various types of coordination mechanisms. Different types of buses can be mixed together in a particular network. Consequently, systems with heterogeneous coordination mech-

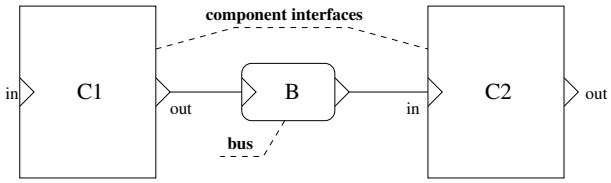

Figure 1: A network of components *C1* and *C2* connected to bus *B*

anisms can be effectively specified using a single uniform formalism. In VCD there is a concept of *bus classes*, which allows to specify generic templates for various coordination media.

The key concept of VCD is in its hierarchical network structure, which unfolds the *coordination layer*. This is achieved by the possibility of taking networks as components of other networks (higher-level networks). The relation between a network and its enclosing interface is defined by a *gate*. Gate maps ports of the lower-level network to ports of the enclosing interface in the higher-level network. An example of a network hierarchy is given in Fig. 2.

At the bottom-most level, behavior of system components has to be specified explicitly. There is no direct visual notation for behavioral specification in VCD. Instead of that, the formal semantic framework of so called VCD *leaves* is defined. It is called *behavioral layer*. The behavioral layer is based on the semantic model given by the notion of input/output labeled transition system (LTS) with sets of input and output actions taken as labels. This allows any language with semantics defined in the domain of LTS to be used for behavioral specification of system components. This property makes VCD heterogeneous also at the behavioral layer. Heterogeneity at this level is achieved with respect to the set of semantically compatible, but notationally different languages, which can be incorporated to VCD for the purpose of behavioral description. As examples of supported languages variants of Statecharts or Petri-Nets can be mentioned.

Semantics of VCD is based on a state transition model. By traversing the network hierarchy, it relies on a formal mechanism of combining component state transition models into one resulting state transition model of the top-most network. This is done with respect to the communication relationships specified by buses. Semantics of a particular bus class represents behavior of a specific communication media.

3 Syntax and Semantics of VCD

In this section, the formal syntax of VCD is defined and its semantics given.

3.1 Syntax

VCD networks are formally represented as VCD *terms*. Before capturing them formally we will build some basic notation.

3.1.1 Ports and Interfaces

The most basic elements of the coordination layer are *interfaces* with *ports*. We fix $\mathcal{W}$ a countable set of *write ports* and $\mathcal{R}$ a countable set of *read ports*. Interface I is defined as a pair consisting of a finite set of input ports and a finite set of output ports — $I = \langle W, R \rangle$, $W \subseteq \mathcal{W}$, $R \subseteq \mathcal{R}$, $W \cap R = \emptyset$. We mark projections $I^W = W$ the *write-interface*, $I^R = R$ the *read-interface*, respectively.

3.1.2 Buses and Bus Classes

The key construct of the coordination layer is *bus*. As it has been mentioned in the previous section, buses represent coordination mechanisms. Particular types of coordination mechanisms are represented as *bus classes*, which are formally defined as input/output labeled transition systems (I/O LTS).

Definition 3.1. Bus class $\mathcal{B}$ *is a tuple* $\langle Q, T, q_0 \rangle$ *where*

- Q *is a finite set of states,*
- $q_0 \in Q$ *an initial state,*
- $T \subseteq Q \times 2^{\mathcal{W}} \times 2^{\mathcal{R}} \times Q$ *a (countable) transition relation.*

Any bus class can be instantiated as a particular bus and used for specification of concrete connections among components in a network. The bus interface is determined by the set of links which connect the bus to the ports of surrounding components. The finiteness of the bus interface puts a finite bound to the transition relation of a bus instance. Formal definition of bus instance, given by its interface and its class, is the following.

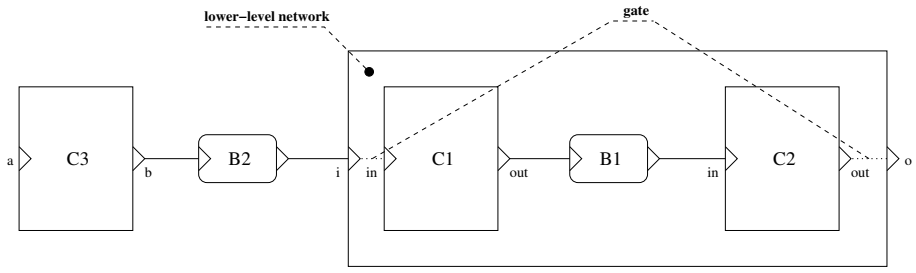

Figure 2: Network hierarchy

Definition 3.2. Bus instance $\mathtt{B}$ *of a bus class* $\mathcal{B}$ *is a tuple* $\mathtt{B} = \langle \mathrm{I}, \mathcal{B} \rangle$*, where* I *is an interface and* $\mathcal{B}$ *a bus class.*

The interface of the bus instance $\mathtt{B}$ will be denoted as $\mathrm{I}(\mathtt{B})$.

3.1.3 Gates, Networks and Leaves

Now we are going to define terms which formally represent VCD network diagrams. In the network depicted in Fig. 2 there are dashed lines which connect ports of subsystem interfaces to ports of the surrounding network interface. Later on in this subsection, these dashed links will be formalized as the notion of *gate*.

Definition 3.3. *A* VCD term *is:*

1. VCD leaf – *behavioral model specified in any LTS-compatible language*
2. VCD network $\mathrm{N} = \langle \bar{\mathrm{C}}, \bar{\mathrm{M}}, \mathrm{L} \rangle$*, where*
   (a) $\bar{\mathrm{C}} = \langle \mathrm{C}_1, \ldots, \mathrm{C}_n \rangle$ – *vector of components*
   (b) $\forall \mathrm{i} : \mathrm{C}_\mathrm{i} = \langle \mathrm{S}_\mathrm{i}, \mathrm{I}_\mathrm{i}, \mathrm{G}_\mathrm{i} \rangle$
      - S_i *... VCD term*
      - I_i *... interface*
      - G_i *... gate (see definition below)*
   (c) $\bar{\mathrm{M}} = \langle \mathrm{M}_1, \ldots, \mathrm{M}_k \rangle$ – *vector of busses*
   (d) $\forall \mathrm{j} : \mathrm{M}_\mathrm{j} = \langle \mathrm{I}_\mathrm{j}, \mathcal{B}_\mathrm{j} \rangle$
      - I_j *... interface of a bus* M_j
      - $\mathcal{B}_\mathrm{j}$ *... class of a bus* M_j

(e) $L \subseteq (\{1,\ldots,n\} \times (\mathcal{W} \cup \mathcal{R})) \times (\{1,\ldots,k\} \times (\mathcal{W} \cup \mathcal{R}))$ *a set of* links *satisfying: if* $\langle\langle i, p_1\rangle, \langle j, p_2\rangle\rangle \in L$ *then:*

i. $p_1 \in \mathcal{W} \Leftrightarrow p_2 \in \mathcal{R}$

ii. $p_1 \in I_i^{\mathcal{W}} \cup I_i^{R}$

iii. $p_2 \in I^{\mathcal{W}}(M_j) \cup I^{R}(M_j)$

iv. $\langle\langle l, p_1'\rangle, \langle j, p_2\rangle\rangle \in L \Leftrightarrow l = i \wedge p_1' = p_1$

v. $\langle\langle i, p_1\rangle, \langle l, p_2'\rangle\rangle \in L \Leftrightarrow l = j \wedge p_2' = p_2$

The set of all VCD terms will be denoted by $\mathcal{S}$.

To formalize the feature of embedding a network into a higher-level network, we set up a function ϵ_R ($\epsilon_{\mathcal{W}}$) which for any VCD network returns a set of all its read (write) ports which have no connection to any bus. We call such ports *free ports*. To overcome ambiguity of port names in the context of a network, we index all the component interfaces in the scope of a particular network, and mark each port with the index of its interface.

Definition 3.4. *Let* $N = \langle \bar{C}, \bar{M}, L\rangle$ *be a network.*

- $\epsilon_{\mathcal{W}}(N) = \{\langle i, w\rangle \mid w \in I_i^{\mathcal{W}} \wedge \forall j, w' : \langle\langle i, w\rangle, \langle j, w'\rangle\rangle \notin L\}$
- $\epsilon_R(N) = \{\langle i, r\rangle \mid r \in I_i^{R} \wedge \forall j, r' : \langle\langle i, r\rangle, \langle j, r'\rangle\rangle \notin L\}$

We define *interface* of network N as a pair $I(N) = \langle \epsilon_{\mathcal{W}}(N), \epsilon_R(N)\rangle$.

Gate is formally defined as a partial function relating ports of a particular component interface to free ports of the network which is nested in that interface. In the case when the nested structure is a leaf, the gate maps interface ports to eponymous actions of the nested process.

Definition 3.5. *Let* I *be an interface.*

1. *Let S be a VCD leaf encapsulated in the interface* I. *Let* $\mathtt{ports}(S) \subseteq \mathcal{W} \cup \mathcal{R}$ *be a set of all actions of S. We define a* gate *of the leaf S as the identity function* $G : I^{\mathcal{W}} \cup I^{R} \to \mathtt{ports}(S)$, $\forall x \in I^{\mathcal{W}} \cup I^{R}.\, G(x) = x$.

2. *Let* $S = \langle\langle\langle S_1, I_1, G_1\rangle, ..., \langle S_n, I_n, G_n\rangle\rangle, \langle M_1, ..., M_k\rangle, L\rangle$ *be a VCD network embedded in interface* I. *We define a* gate *of the network S as the partial function* $G : I^{\mathcal{W}} \cup I^{R} \to I(S)$ *satisfying:*

   - $\forall w \in I^{\mathcal{W}}.\, G(w) = \langle i, w'\rangle \wedge \langle i, w'\rangle \in \epsilon_{\mathcal{W}}(S)$
   - $\forall r \in I^{R}.\, G(r) = \langle i, r'\rangle \wedge \langle i, r'\rangle \in \epsilon_R(S)$

3.2 Semantics

Before we present the formal semantics of VCD, we establish some notation. Let N be a network containing just $n > 0$ components. Further let I_i be the interface of the ith component of N and $\Gamma \subseteq I_i$ some set of its ports. We will denote $\langle i, \Gamma \rangle = \{\langle i, w \rangle \mid w \in \Gamma\}$ the set of ports indexed by the ith component in the network N. Note that if $\Gamma = \emptyset$ then also $\langle i, \Gamma \rangle = \emptyset$.

As a semantic domain a class $\mathcal{L}$ of input/output labeled transition systems (I/O LTS) with sets of input and output actions in transition labels is used. Formally, the semantics is defined as a mapping $\psi : \mathcal{S} \to \mathcal{L}$ which assigns an I/O LTS to each VCD term.

First of all, we define the notion of I/O LTS, which makes the semantic domain for both the behavioral and the coordination layer.

Definition 3.6. *An* I/O LTS *is a tuple* $\langle Q, T, q_0 \rangle$ *where*

- Q *is a finite set of states,*
- $q_0 \in Q$ *an initial state,*
- $T \subseteq Q \times 2^{\mathcal{R}} \times 2^{\mathcal{W}} \times Q$ *a transition relation.*

At the behavioral layer, the state transition semantics captures the dynamics of atomic components. As VCD does not include any predefined syntactic construct for the behavioral layer, this I/O LTS is the structure in which the formalisms for behavioral description have to be encoded.

At the coordination layer, the semantics of a VCD network is defined as a global I/O LTS which composes transitions of local I/O LTSs representing the semantics of network components. This composition is realized with respect to the coordination model given by the specific bus classes instantiated in the network. States of the global I/O LTS are represented as *network configurations*. They respect the hierarchical structure of network terms. The formal definition of a network configuration is the following.

Definition 3.7. *Let* $N = \langle\langle C_1, ..., C_n \rangle, \langle M_1, ..., M_k \rangle, L\rangle$ *be a network. We define its* configuration $\langle \bar{s}, \bar{b} \rangle$ *as a vector of component and bus states* $\langle\langle s_1, ..., s_n \rangle, \langle b_1, ..., b_n \rangle\rangle$ *where* $\forall i \in \{1, ..., n\}.\, s_i$ *is a state of a component* C_i *and* $\forall j \in \{1, ..., k\}.\, b_j$ *is a state of a bus* M_j.

A network configuration contains a vector of current states of components and a vector of current states of buses. Such network configurations determine states of the

resulting I/O LTS. Transitions of the global I/O LTS are defined with respect to the network hierarchy using Plotkin-style inference rules.

Formally, let $N = \langle \bar{C}, \bar{M}, L \rangle$ be a network term. We define its semantics $\psi(N)$ as an I/O LTS $\psi(N) = \langle Q_N, T_N, Q_{0_N} \rangle \in \mathcal{L}$ in which:

- The set of states Q_N is given by all the network configurations.
- Q_{0_N} is a set of initial states – these are the configurations in which at least one of the substates is initial state of some network component.
- The transition relation $T_N \subseteq Q_N \times 2^{\mathcal{N}\times\mathcal{R}} \times 2^{\mathcal{N}\times\mathcal{W}} \times Q_N$ is defined using Plotkin-style inference rules, which combine transitions of subsystems with respect to the network hierarchy.

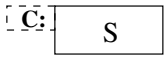

Figure 3: Subsystem S embedded in a component C

In figure 3, there is a scheme how the component C is built by embedding of the subsystem S into a component interface. The subsystem S can be either a leaf or a network. Transition system of the component C is derived from the transition system of the embedded subsystem with respect to ports in the interface. Actions of S which have no ports in the interface of component C are hidden. The structure of the relevant inference rule for the situation when S is a leaf is the following.

Let $C = \langle S, I, G \rangle$ be a component of the network N. We suppose that T_S is a transition relation of the VCD term S. We define the transition relation $T_C \subseteq Q_C \times 2^{I^R} \times 2^{I^W} \times Q_C$ of the network component C. It is derived from T_S with respect to the interface I and the gate G. There are two cases of which type the subsystem S can be. With respect to this situation, the derivation of T_C from T_S is defined by one of the following inference rules.

1. In the case when S is a VCD leave, $S = \langle Q_S, T_S, q_{0_S} \rangle$, the transition relation T_C is derived directly from T_S as stated in the following rule:

$$\frac{T_S: \quad q \xrightarrow[\Delta]{\Gamma} q' \quad (\Gamma \subseteq ports(S) \cap \mathcal{R} \text{ and } \Delta \subseteq ports(S) \cap \mathcal{W})}{T_C: \quad q \xrightarrow[I^W \cap \Delta]{I^R \cap \Gamma} q'}$$

The only difference between T_S and T_C is that events of T_S which are not in the component interface are abstracted in T_C by deleting them. Note that this rule also lifts internal leaf $q \xrightarrow[\emptyset]{\emptyset} q'$ transitions to internal component transitions.

2. For $S = \langle \bar{C}, \bar{M}, L \rangle$ a network term we have the rule:

$$\frac{T_S : \quad \langle \bar{s}, \bar{b} \rangle \xrightarrow[\Delta^\times]{\Gamma^\times} \langle \bar{s}', \bar{b}' \rangle}{T_C : \quad \langle \bar{s}, \bar{b} \rangle \xrightarrow[G^{-1}(\Delta^\times)]{G^{-1}(\Gamma^\times)} \langle \bar{s}[i := q_i'], \bar{b} \rangle}$$

Notation $\Gamma^\times \subseteq \{\langle i, w \rangle \,|\, i \in \mathcal{N}, w \in \mathcal{W}\}$ denotes a set of indexed input events. Similarly, $\Delta^\times \subseteq \{\langle i, r \rangle \,|\, i \in \mathcal{N}, r \in \mathcal{R}\}$ denotes a set of indexed output events. $G^{-1}(\Gamma^\times)$ stands for the set of ports in network interface I with which events in $\Gamma^\times$ are related by the gate G. Analogously, similar notation is also used for the indexed output events $\Delta^\times$. $\bar{s}[i := q_i']$ is the state vector which was constructed from $\bar{s}$ by replacing its ith component with the state q_i'.

In the same way like the previous rule, this rule also propagates the internal events and abstracts from those events of the network S which are not assigned to any port of the interface I.

Now we are going to establish inference rules which define the transition relation T_N of network configurations. It will be derived from the component transition relations T_{C_i} and the transitions of buses. The key feature of these rules is building of network configurations (global state vectors) from component configurations (local state vectors).

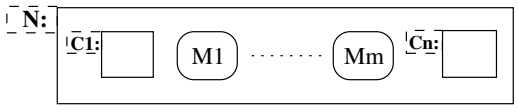

Figure 4: Components $C_1 \ldots C_n$ and buses $M_1 \ldots M_m$ embedded in a network N

In figure 4, there is a scheme of a network with n components arbitrarily connected to m buses. For simplification, the links are not depicted. To define a global transition system for the network N, transition systems of the components and buses have to be composed. There are two different situations:

- Stand-alone components — their transitions are interleaved.
- Components connected to buses — their transitions are interleaved or synchronized w.r.t. semantics of instantiated bus classes.

To resolve the first situation, we add to T_N all the component transitions which are totally independent of any bus interconnections. The following rule defines interleaving behavior of components in the network N.

$$3.\ \frac{T_{C_i}: \quad \bar{s}[i] \xrightarrow[\Delta]{\Gamma} q_i' \quad \langle i,\Gamma\rangle \subseteq \epsilon_R(N), \langle i,\Delta\rangle \subseteq \epsilon_W(N)}{T_N: \quad \langle \bar{s}, \bar{b}\rangle \xrightarrow[\langle i,\Delta\rangle]{\langle i,\Gamma\rangle} \langle \bar{s}[i := q_i'], \bar{b}\rangle}$$

Notation $\bar{s}[i]$ denotes the state configuration of the ith component of N. Note that internal component events are lifted by this rule too.

Finally, we are approaching to the last inference rule, which is the most complex one. It puts together transitions of buses and transitions of components and evaluates their relationships given by the network links. According to the evaluated result it can then coordinate some components by firing their transitions synchronously with transitions of some buses. Before we will define such a coordination rule, we have to look deeper into the structure of the network.

Let $N = \langle\langle C_1, ..., C_n\rangle, \langle M_1, ..., M_m\rangle, L\rangle$ be a network for some $m, n \in \mathcal{N}$. With respect to the link relation L some strongly connected blocks of components may be distinguished in the network. For each such a block of components we will define a synchronizing coordination rule. From the semantical point of view, any such a separated block of components can internally synchronize while different blocks put together may only mutually interleave. In other words, these blocks are the maximal groups of components with potential synchronous behavior.

To capture the partitioning idea formally, we define a relation R, $R \subseteq \{1, ..., n\} \times \{1, ..., n\}$:

$$\langle i, j\rangle \in R \overset{df}{\Leftrightarrow} i = j \vee \exists k \in \{1, ..., m\}, p_i \in I_i, p_k \in I(M_k), p_j \in I_j.$$
$$\langle\langle i, p_i\rangle, \langle k, p_k\rangle\rangle \in L \wedge \langle\langle j, p_j\rangle, \langle k, p_k\rangle\rangle \in L$$

It is worth noting that R is an equivalence. We will note $\{1, ..., n\}_{|R} \subseteq 2^{\{1,...,n\}}$ set of all classes of equivalence over the set of component indexes $\{1, ..., n\}$.

Let $\Omega \in \{1, ..., n\}_{|R}$. We will denote $\Omega' \subseteq \{1, ..., m\}$ a set of indexes of buses which are connected to components indexed by Ω. Precisely,

$$\Omega' = \{i \in \{1, ..., m\} \mid \exists k \in \Omega, p_k \in I_k, p_i \in I(M_i).\ \langle\langle k, p_k\rangle, \langle i, p_i\rangle\rangle \in L\}.$$

Now let $q \equiv \langle \bar{s}, \bar{b} \rangle$ be an actual configuration of network N. We define sets $ET_\Omega(q)$ and $ET_{\Omega'}(q)$ of all transitions starting in q and indexed by their component (respectively bus) indexes:

$$ET_\Omega(q) = \{\langle i, t\rangle \mid \forall i \in \Omega.\, t \in T_{C_i},\ src(t) = \bar{s}[i]\}$$
$$ET_{\Omega'}(q) = \{\langle i, t\rangle \mid \forall i \in \Omega', t \in T(M_i).\, src(t) = \bar{b}[i]\}$$

The notation $src(t)$ denotes the source state of the transition t and $T(M_i)$ denotes the transition relation of the bus M_i.

To precisely characterize the set of all component transitions which can be synchronized with buses resulting in the one global network transition, we have to put some constraints on $ET_\Omega(q)$ and $ET_{\Omega'}(q)$. Firstly, we require that for each source state only one transition is included. Formally, we say $ET_\Omega(q)$ is *consistent* if and only if $\forall i, j, t, t'.\, \langle i, t\rangle \in ET_\Omega(q) \wedge \langle j, t'\rangle \in ET_\Omega(q) \Rightarrow i \neq j$.

Further we define a triggering relation among component and bus transitions of a particular partition of current network configuration. Firstly we extract some sets of events from the sets of component (bus) transitions $ET_\Omega(q)$ and $ET_{\Omega'}(q)$. In the following definitions, the notations $\Delta(t)$ and $\Gamma(t)$ denote all the output (input) events which occur in the label of the transition t.

- $\mathcal{E}_\Delta(\Omega) = \{\langle i, w\rangle \mid \exists \langle i, t\rangle \in ET_\Omega(q).w \in \Delta(t) \wedge \langle i, w\rangle \in \epsilon_W(N)\}$
- $\mathcal{E}_\Gamma(\Omega) = \{\langle i, r\rangle \mid \exists \langle i, t\rangle \in ET_\Omega(q).r \in \Gamma(t) \wedge \langle i, r\rangle \in \epsilon_R(N)\}$
- $\mathcal{F}_\Delta(\Omega) = \{\langle j, w'\rangle \mid \exists i \in \Omega, w \in \mathcal{W}, \langle i, t\rangle \in ET_\Omega(q).\, \langle\langle i, w\rangle, \langle j, w'\rangle\rangle \in L \wedge w \in \Delta(t)\}$
- $\mathcal{F}_\Gamma(\Omega) = \{\langle j, r'\rangle \mid \exists i \in \Gamma, r \in \mathcal{R}, \langle i, t\rangle \in ET_\Omega(q).\, \langle\langle i, r\rangle, \langle j, r'\rangle\rangle \in L \wedge r \in \Gamma(t)\}$
- $\mathcal{A}_\Delta(\Omega') = \{\langle i, w\rangle \mid \exists \langle i, t\rangle \in ET_{\Omega'}(q).\, w \in \Delta(t)\}$
- $\mathcal{A}_\Gamma(\Omega') = \{\langle i, w\rangle \mid \exists \langle i, t\rangle \in ET_{\Omega'}(q).\, w \in \Gamma(t)\}$

We say $ET_\Omega(q)$ *triggers* $ET_{\Omega'}(q)$ iff the following two conditions hold:

1. $\mathcal{A}_\Gamma(\Omega') = \mathcal{F}_\Delta(\Omega)$
2. $\mathcal{A}_\Delta(\Omega') = \mathcal{F}_\Gamma(\Omega)$

For each partition Ω we now define the final coordination rule:

$$4.\ \frac{ET_{\Omega}(\langle \bar{s}, \bar{b} \rangle) \text{ and } ET_{\Omega'}(\langle \bar{s}, \bar{b} \rangle) \text{ consistent, } ET_{\Omega}(\langle \bar{s}, \bar{b} \rangle) \text{ triggers } ET_{\Omega'}(\langle \bar{s}, \bar{b} \rangle)}{T_N : \quad \langle \bar{s}, \bar{b} \rangle \xrightarrow[\varepsilon_{\Delta}(\Omega)]{\varepsilon_{\Gamma}(\Omega)} \langle \bar{s}', \bar{b}' \rangle}$$

where:

$$\bar{s}'[i] = \begin{cases} s_i', \text{ if } \exists t \in T_{C_i}.\, t \in ET_{\Omega}(\langle \bar{s}, \bar{b} \rangle) \text{ so that } trg(t) = s_i' \\ \bar{s}[i], \text{ otherwise} \end{cases}$$

$$\bar{b}'[i] = \begin{cases} b_i', \text{ if } \exists t \in T(M_i).\, t \in ET_{\Omega'}(\langle \bar{s}, \bar{b} \rangle) \text{ so that } trg(t) = b_i' \\ \bar{b}[i], \text{ otherwise} \end{cases}$$

The notation $trg(t)$ denotes the target state of the transition t.

4 Behavioral Equivalence of VCD Specifications and Architectural Compatibility

In the last section the notion of VCD terms was defined and its semantics was given in structural operational manner. In this section, observational equivalence relation on the set of VCD terms is defined. Finally, the notion of architectural compatibility of VCD specifications is established.

4.1 Weak bisimulation

At first, we define the notion of weak bisimulation equivalence on states of VCD leaves. Afterwards, we will extend it to deal with VCD network terms.

Note that according to the semantics of VCD components IOLTSs can perform internal transitions in the form $\xrightarrow[\emptyset]{\emptyset}$. These transitions can appear as consequences of several situations. Firstly, they can be a result of network level synchronization. Secondly, they can be implicit in behavior of VCD leaves. Lastly, they can appear in consequence of so called hiding when the network or a VCD leaf is inserted into an interface which does not include ports for all events contained in subsystem's transitions. To refine the notion of equivalence to be insensitive to this kind of events we define the weak (observational) bisimulation equivalence.

Definition 4.1. *Let* $\langle Q, T, q \rangle$ *be a configuration of an IOLTS. Further let* $t \in (2^{\mathcal{R}} \times 2^{\mathcal{W}})^*$, $t = \langle \Gamma_1, \Delta_1 \rangle \langle \Gamma_2, \Delta_2 \rangle \cdots \langle \Gamma_n, \Delta_n \rangle$, $n \geq 0$ *be sequence of read/write event sets of some trace such*

that $\exists q_1, ..., q_n \in Q.\, q \xrightarrow[\Delta_1]{\Gamma_1} q_1 \xrightarrow[\Delta_2]{\Gamma_2} \cdots \xrightarrow[\Delta_n]{\Gamma_n} q_n$. *To shorten our notation we denote this situation simply as* $q \xrightarrow{t} q_n$.

1. *We define* $\hat{t} \in (2^{\mathcal{R}} \setminus \{\emptyset\} \times 2^{\mathcal{W}} \setminus \{\emptyset\})^*$ *a trace sequence taken from* t *by deleting all occurrences of subsequences of the form* $\langle \emptyset, \emptyset \rangle$ *occurring at any position of the trace* t.

2. *Let* $\hat{t} \in (2^{\mathcal{R}} \setminus \{\emptyset\} \times 2^{\mathcal{W}} \setminus \{\emptyset\})^*$. *We write* $q \overset{t}{\Rightarrow} q'$ *if there exists* t' *such that* $q \xrightarrow{t'} q'$ *and* $t = \hat{t'}$. *We call the transition* $\overset{t}{\Rightarrow}$ *the* weak *transition.*

In the following definition we use the notation $\langle Q, T, q_1 \rangle$ to represent a configuration of an IOLTS which is currently in the state q_1. Respective VCD leave term can be taken then as a special case of this notation when the term represents an initial configuration of the IOLTS.

Now we define the weak bisimulation equivalence for configurations of VCD leaves.

Definition 4.2. *Let* $\langle Q_1, T_1, q_1 \rangle$ *and* $\langle Q_2, T_2, q_2 \rangle$ *be VCD leaves. We say they are* weakly (observationally) bisimilar *(*$\langle Q_1, T_1, q_1 \rangle \approx \langle Q_2, T_2, q_2 \rangle$*) if and only if there exists a relation* $R \subseteq Q_1 \times Q_2$ *such that* $(q_1, q_2) \in R$ *satisfying:*
if $(q_1, q_2) \in R$ *then:*

1. $q_1 \xrightarrow[\Delta]{\Gamma} q_1' \in T_1 \Rightarrow \exists q_2'.\, q_2 \overset{\hat{t}}{\Rightarrow} q_2' \in T_2$ *and* $(q_1', q_2') \in R$

2. $q_2 \xrightarrow[\Delta]{\Gamma} q_2' \in T_2 \Rightarrow \exists q_1'.\, q_1 \overset{\hat{t}}{\Rightarrow} q_1' \in T_1$ *and* $(q_1', q_2') \in R$ *where* $t = \langle \Gamma, \Delta \rangle$.

We aim to extend the definition of weak bisimulation equivalence to network terms. Recall from the last section that the transition relation over network terms expects indexed sets of input and output events. The following definition extends the notion of weak transition to deal with indexed input and output event sets.

Definition 4.3. *Let* $\langle Q, T, q \rangle$ *be a configuration of an IOLTS. Further let* $t \in ((\mathcal{N} \times 2^{\mathcal{R}}) \times (\mathcal{N} \times 2^{\mathcal{W}}))^*$, $t = \langle \Gamma_1^\times, \Delta_1^\times \rangle \langle \Gamma_2^\times, \Delta_2^\times \rangle \cdots \langle \Gamma_n^\times, \Delta_n^\times \rangle$, $n \geq 0$ *be sequence of read/write event sets of some trace such that* $\exists q_1, ..., q_n \in Q.\, q \xrightarrow[\Delta_1^\times]{\Gamma_1^\times} q_1 \xrightarrow[\Delta_2^\times]{\Gamma_2^\times} \cdots \xrightarrow[\Delta_n^\times]{\Gamma_n^\times} q_n$. *To shorten our notation we denote this situation simply as* $q \xrightarrow{t} q_n$.

1. *We define* $\hat{t} \in ((\mathcal{N} \times 2^{\mathcal{R}} \setminus \{\emptyset\}) \times (\mathcal{N} \times 2^{\mathcal{W}} \setminus \{\emptyset\}))^*$ *a trace sequence taken from* t *by deleting all occurrences of subsequences of the form* $\langle \emptyset, \emptyset \rangle$ *occurring at any position of the trace* t.

2. *Let* $\hat{t} \in ((\mathcal{N} \times 2^{\mathcal{R}} \setminus \{\emptyset\}) \times (\mathcal{N} \times 2^{\mathcal{W}} \setminus \{\emptyset\}))^*$. *We write* $q \overset{t}{\Rightarrow} q'$ *if there exists* t' *such that* $q \xrightarrow{t'} q'$ *and* $t = \hat{t'}$.

Finally, we extend the definition of weak bisimulation equivalence to the set of network configurations.

Definition 4.4. *Let* $\langle \bar{s}, \bar{b} \rangle = \langle\langle s_1, ..., s_n \rangle, \langle b_1, ..., b_m \rangle\rangle$ *and* $\langle \bar{s'}, \bar{b'} \rangle = \langle\langle s'_1, ..., s'_{n'} \rangle, \langle b'_1, ..., b'_{m'} \rangle\rangle$ *be network configurations, where* $m, m', n, n' \geq 1$. *We say they are* (weakly) bisimilar *(*$\langle \bar{s}, \bar{b} \rangle \approx \langle \bar{s'}, \bar{b'} \rangle$*) if and only if there exists a relation* $R \subseteq ((S_1 \times ... \times S_n) \times (M_1 \times ... \times M_m)) \times ((S'_1 \times ... \times S'_{n'}) \times (M'_1 \times ... \times M'_{m'}))$ *such that* $(\langle \bar{s}, \bar{b} \rangle, \langle \bar{s'}, \bar{b'} \rangle) \in R$ *satisfying: if* $(q_1, q_2) \in R$ *then:*

1. $q_1 \xrightarrow[\Delta^\times]{\Gamma^\times} q'_1 \in T_1 \Rightarrow \exists q'_2.\ q_2 \stackrel{\hat{t}}{\Rightarrow} q'_2 \in T_2$ *and* $(q'_1, q'_2) \in R$
2. $q_2 \xrightarrow[\Delta^\times]{\Gamma^\times} q'_2 \in T_2 \Rightarrow \exists q'_1.\ q_1 \stackrel{\hat{t}}{\Rightarrow} q'_1 \in T_1$ *and* $(q'_1, q'_2) \in R$ *where* $t = \langle \Gamma^\times, \Delta^\times \rangle$.

To set up a congruence based on the weak bisimulation, we have to slightly modify the definition of weak bisimulation to establish an observational congruence.

Definition 4.5. *Let* $\langle Q_1, T_1, q_1 \rangle$ *and* $\langle Q_2, T_2, q_2 \rangle$ *be VCD leaves. We say they are* weakly (observationally) congruent *(*$\langle Q_1, T_1, q_1 \rangle \approx_C \langle Q_2, T_2, q_2 \rangle$*) if and only if there exists a relation* $R \subseteq Q_1 \times Q_2$ *so that* $(q_1, q_2) \in R$ *and satisfying:*
if $(q_1, q_2) \in R$ *then:*

1. $q_1 \xrightarrow[\Delta]{\Gamma} q'_1 \in T_1 \Rightarrow \exists q'_2.\ q_2 \stackrel{t}{\Rightarrow} q'_2 \in T_2$ *and* $q'_1 \approx q'_2$
2. $q_2 \xrightarrow[\Delta]{\Gamma} q'_2 \in T_2 \Rightarrow \exists q'_1.\ q_1 \stackrel{t}{\Rightarrow} q'_1 \in T_1$ *and* $q'_1 \approx q'_2$*) where* $t = \langle \Gamma, \Delta \rangle$.

The observational congruence can be extended to network terms in the same way like the weak bisimulation.

Now we can state a theorem concerning the required congruence property for our newly setup equivalence.

Theorem 4.6. *Let* $N = \langle\langle C_1, ..., C_i, ..., C_n \rangle, \langle M_1, ..., M_m \rangle, L\rangle$ *and* $N' = \langle\langle C_1, ..., C'_i, ..., C_n \rangle, \langle M_1, ..., M_m \rangle, L\rangle$ *be VCD network terms which differs only in the* i*th component. Further let* S_i *and* S'_i *be arbitrary VCD terms for some* $0 < i \leq n$*, satisfying* $C_i = \langle S_i, I_i, G_i \rangle$ *and* $C'_i = \langle S'_i, I_i, G_i \rangle$ *for some interface* I_i *and gate* G_i. *Then the following holds:*

If $S_i \approx_C S'_i$ *then also* $N \approx_C N'$.

Proof. By induction w.r.t. structure of configurations of VCD terms.

We assume $S_i \approx_C S'_i$.

Let $q_N = \langle \bar{s}, \bar{b} \rangle$ be an arbitrary configuration of network N. There are several possibilities for the transition $q_N \xrightarrow[\Delta^\times]{\Gamma^\times} q'_N$ for some $\Gamma \subset \mathcal{R}$, $\Delta \subset \mathcal{W}$.

1. $q_N = \langle\langle s_1, ..., s_j, ..., s_n\rangle, \langle b_1, ..., b_m\rangle\rangle$
   $q'_N = \langle\langle s_1, ..., s'_j, ..., s_n\rangle, \langle b_1, ..., b_m\rangle\rangle$, where $0 < j \leq n$ and $j \neq i$

   The only way in which this network transition could happen is the inference rule (3):

$$\frac{T_{C_j}: \quad \bar{s}[j] \xrightarrow[\Delta]{\Gamma} q'_j \quad \langle j, \Gamma\rangle \subseteq \epsilon_R(N), \langle j, \Delta\rangle \subseteq \epsilon_W(N)}{T_N: \quad q_N \xrightarrow[\langle j,\Delta\rangle]{\langle j,\Gamma\rangle} \langle \bar{s}[j := q'_j], \bar{b}\rangle}$$

   Due to fact that C_j is a component contained also in the network N', the same transition is enabled also in the configuration $q_{N'}$ of the network N':

$$\frac{T_{C_j}: \quad \bar{s}[j] \xrightarrow[\Delta]{\Gamma} q'_j \quad \langle j, \Gamma\rangle \subseteq \epsilon_R(N), \langle j, \Delta\rangle \subseteq \epsilon_W(N)}{T_{N'}: \quad q_{N'} \xrightarrow[\langle j,\Delta\rangle]{\langle j,\Gamma\rangle} \langle \bar{s}[j := q'_j], \bar{b}\rangle}$$

   In consequence, both configurations q_N and $q_{N'}$ lead by the transition $\xrightarrow[\Delta^\times]{\Gamma^\times}$ to the structurally simpler resulting network configurations q'_N, $q'_{N'}$. Applying the induction assumption, we have $q'_N \approx q'_{N'}$.

2. $q_N = \langle\langle s_1, ..., s_i, ..., s_n\rangle, \langle b_1, ..., b_m\rangle\rangle$
   $q'_N = \langle\langle s_1, ..., s'_i, ..., s_n\rangle, \langle b_1, ..., b_m\rangle\rangle$

   Analogously to the previous situation, the inference rule (3) is the only one which could cause the transition from q_N to q'_N.

$$\frac{T_{C_i}: \quad \bar{s}[i] \xrightarrow[\Delta]{\Gamma} q'_i \quad \langle i, \Gamma\rangle \subseteq \epsilon_R(N), \langle i, \Delta\rangle \subseteq \epsilon_W(N)}{T_N: \quad q_N \xrightarrow[\langle i,\Delta\rangle]{\langle i,\Gamma\rangle} \langle \bar{s}[i := q'_i], \bar{b}\rangle}$$

   In contrary to the previous case, N differs from N' just in the ith component in which the investigated transition takes place. We have to evolve this transition with respect to one of the inference rules (1) or (2). It depends on the form of the S_i term, which one has to be chosen.

   Suppose S_i is a leaf. Then the following rule is applied:

$$\frac{T_{S_i}: \quad q_{S_i} \xrightarrow[\Delta]{\Gamma} q'_{S_i} \quad (\Gamma \subseteq ports(S_i) \cap \mathcal{R} \text{ and } \Delta \subseteq ports(S_i) \cap \mathcal{W})}{T_{C_i}: \quad q_{C_i} \xrightarrow[I^W \cap \Delta]{I^R \cap \Gamma} q'_{C_i}}$$

We know that $S_i \approx_C S'_i$. In consequence, there exists a transition $q_{S'_i} \stackrel{\hat{t}}{\Rightarrow} q'_{S'_i}$, $t = \langle \Gamma, \Delta \rangle$ such that $q'_{S'_i} \approx q'_{S_i}$. The following rule can be applied now to infer a weak transition of the component C'_i:

$$\frac{T_{S'_i} : \quad q_{S'_i} \stackrel{\hat{t}}{\Rightarrow} q'_{S'_i}}{T_{C'_i} : \quad q_{C'_i} \stackrel{\hat{t'}}{\Rightarrow} q'_{C'_i}}$$

where $t' = \langle I^R \cap \Gamma, I^W \cap \Delta \rangle$. Note that both components C_i and C'_i have the same interface I.

Now the following weak transition of the network configuration $q_{N'}$ can be inferred:

$$\frac{T_{C'_i} : \quad \bar{s}[i] \stackrel{\hat{t'}}{\Rightarrow} q'_i \quad \langle i, \Gamma \rangle \subseteq \epsilon_R(N), \langle i, \Delta \rangle \subseteq \epsilon_W(N)}{T_{N'} : \quad q_{N'} \stackrel{\hat{t''}}{\Rightarrow} \langle \bar{s}[i := q'_i], \bar{b} \rangle}$$

where $t'' = \langle \langle i, I^R \cap \Gamma \rangle, \langle i, I^W \cap \Delta \rangle \rangle$.

3. $q_N = \langle \langle s_1, ..., s_i, ..., s_n \rangle, \langle b_1, ..., b_m \rangle \rangle$
   $q'_N = \langle \langle s'_1, ..., s'_i, ..., s'_n \rangle, \langle b'_1, ..., b'_m \rangle \rangle$, where $0 < j \leq n$

   This case deal with the synchronization transition. The only possible way how it could be inferred is with respect to the rule (4):

$$\frac{ET_\Omega(\langle \bar{s}, \bar{b} \rangle) \text{ and } ET_{\Omega'}(\langle \bar{s}, \bar{b} \rangle) \text{ consistent, } ET_\Omega(\langle \bar{s}, \bar{b} \rangle) \text{ triggers } ET_{\Omega'}(\langle \bar{s}, \bar{b} \rangle)}{T_N : \quad \langle \bar{s}, \bar{b} \rangle \xrightarrow[\varepsilon_\Delta(\Omega)]{\varepsilon_\Gamma(\Omega)} \langle \bar{s'}, \bar{b'} \rangle}$$

   There exist several cases how the ith component can influence the synchronization.

   (a) If $i \notin \Omega$ then there is no influence and the network N' can do just the same transitions as the network N from the current configuration q_N. Hence, the result is trivial.

   (b) If $i \in \Omega$ then the transition of component C_i has influence on the synchronization. Let $q_{C_i} \xrightarrow[\Delta]{\Gamma} q'_{C_i}$ be such a transition. The only way how it could be inferred is application of the rule (1) or (2) similarly as in the previous case (for presentation here we choose the situation of a leaf component form):

$$\frac{T_{S_i} : \quad q_{S_i} \xrightarrow[\Delta]{\Gamma} q'_{S_i} \quad (\Gamma \subseteq ports(S_i) \cap \mathcal{R} \text{ and } \Delta \subseteq ports(S_i) \cap \mathcal{W})}{T_{C_i} : \quad q_{C_i} \xrightarrow[I^W \cap \Delta]{I^R \cap \Gamma} q'_{C_i}}$$

We know $S_i \approx_C S'_i$. It implies that there exists a transition $q'_{S'_i} \stackrel{\hat{t}}{\Rightarrow} q'_{S'_i}$, $t = \langle \Gamma, \Delta \rangle$ such that $q'_{S'_i} \approx q'_{S_i}$. The following scheme[1] can be applied now to infer a weak transition of the component C'_i:

$$\frac{T_{S'_i} : \quad q_{S'_i} \stackrel{\hat{t}}{\Rightarrow} q'_{S'_i}}{T_{C'_i} : \quad q_{C'_i} \stackrel{\hat{t'}}{\Rightarrow} q'_{C'_i}}$$

where $t' = \langle I^R \cap \Gamma, I^W \cap \Delta \rangle$.

In consequence, the component C'_i is capable of firing a transition with identical label sets as the transition of the component C_i which influences the synchronization of a network cluster Ω. This fact implies the required result that network N' can fire from its current configuration $q_{N'}$ the same synchronization transition as the network N from its current configuration q_N.

Analogously the symmetric case can be proved to fulfill all the requirements of bisimulation.

□

We defined the notion of weak bisimulation equivalence inspired by the classical work of [13] and we customized it to the setting of VCD terms. In the next subsection, the notion of weak bisimulation equivalence will be used as a base for establishing a framework for checking of architectural compatibility of VCD specifications.

4.2 Architectural compatibility in VCD

In Wright [1], the notion of so called architectural compatibility was established. The question stated there is to ensure formally if a component can safely communicate through a particular connector.

In the setting of VCD diagrams we take this notion in the way of mutual compatibility of components and bus instances in VCD networks. For a particular network we say that it satisfies the property of architectural compatibility if each pair of a component and a bus instance in the network is compatible. Compatibility of a pair of a component and a bus instance is ensured if all possible behavior of a bus instance which is responsible for communication with a component is equivalent to the communication behavior of that component.

[1] This is not a correct rule of VCD semantics, but a shorten form representing a sequence of VCD rules with respect to decomposition of $\Rightarrow$-transitions.

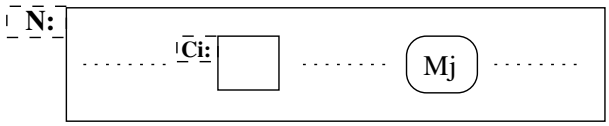

Figure 5: Scheme of compatibility of a component and a bus instance pair

Formally we define compatibility of a component and a bus instance using the notion of weak bisimulation. In Fig. 5 there is a scheme of a network in which we are interested in compatibility of one of its components ($C_i = \langle S_i, I_i, G_i \rangle$) with one of its bus instances ($M_j = \langle I, \mathcal{B} \rangle$, for some bus class $\mathcal{B}$). We suppose that the component C_i has interface $I_i = \langle W_i, R_i \rangle$, $W_i \subset \mathcal{W}$ and $R_i \subset \mathcal{R}$. Further we suppose that there is some nonempty set $P \subseteq W_i \cup R_i$ of ports for which there exist links to a bus instance M_j (so that also $P \subseteq I^W(M_j) \cup I^R(M_j)$).

It is worth noting that the bus instance M_j is semantically represented as an IOLTS. In consequence, the notion of weak bisimulation equivalence can be reasonably established between this IOLTS and an IOLTS S_i which represents the semantics of a component C_i.

To take only the relevant ports (the set P) into account, we create an abstract component C'_i of a component C_i in the following manner.

$$C'_i := \langle S_i, P, G'_i \rangle$$

where

- $G'_i(x) = G_i(x)$, if $x \in P$
- $G'_i(x) = \perp$, otherwise

Similar abstraction has to be applied to the bus instance $M_j = \langle I_j, \mathcal{B} \rangle$. The construction of this abstraction is based on the idea of hiding of all irrelevant transitions of the underlying IOLTS. Formally we define the abstract bus instance M'_j as follows.

Let $\mathcal{L}_j = \langle Q, T, q_0 \rangle$ be the finite state IOLTS which represents the bus instance M_j. We define its abstraction by hiding from its transitions the events which are out of the scope of the set of ports P.

$$\mathcal{L}'_j := \langle Q, T', q_0 \rangle$$

where T' is derived from T according in terms of the following schema:

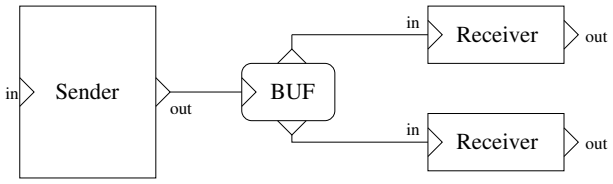

Figure 6: Example of a concrete VCD network

$$\bullet \quad \frac{T: \quad q \xrightarrow[\Gamma]{\Delta} q'}{T': \quad q \xrightarrow[P^R \cap \Gamma]{P^W \cap \Delta} q'}$$

To complete setting of preconditions for definition of the notion of compatibility we have to recall that the semantic function ψ, which has been defined in the section 3.2, returns an IOLTS for any VCD term.

Finally, we say that a component C_i is *architecturally compatible* with a bus instance M_j iff $\psi(C'_i) \approx \mathcal{L}'_j$.

5 An Example of Architectural Specification in VCD

In this section we will demonstrate on a very simple example how the VCD formalism, especially the concept of bus classes, can be used for specification of a distributed software architecture.

In distributed software architectures, components of systems interact most typically in asynchronous way. One of the coordination mechanisms which captures this flavor of interaction is asynchronous message passing. In figure 6 there is a simple VCD network which specifies a distributed system with three components. It can be taken as a part of the specification of some communication protocol. There are one sender and two receiver components in the system. The intended behavior of the sender is to pass the output information to the communication media and continue some inner computation. On the other side, the behavior of any receiver is to take the information from the media asynchronously with computations of the sender.

To model this kind of interaction in the VCD framework, we establish a class $\mathcal{B}_{amp}$ of asynchronous message passing buses. It can be formally defined as the I/O LTS $\mathcal{B}_{amp} = \langle Q, T, q_0 \rangle$ where:

- $Q = \{q_w \mid w \in \mathcal{W}\} \cup q_0$

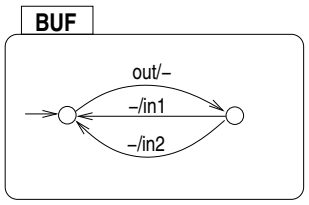

Figure 7: Instance of asynchronous message-passing bus class

- T is defined by disjunction of the following expressions:

  1. $\forall w \in \mathcal{W}.\ \langle q_0, \{w\}, \emptyset, q_w \rangle \in T$
  2. $\forall q_x \in Q.\ \langle q_x, \emptyset, \{x\}, q_0 \rangle \in T$
  3. $\forall q_x \in Q.\ \langle q_x, \emptyset, \emptyset, q_x \rangle \in T$

The first expression defines the reaction of the bus to incoming write-actions. In the second expression, the interaction with receiver components is solved. The last expression adds the empty self-transitions, which allow the interactions to be potentially asynchronous.

The countable transition relation, which is the part of the bus class defined above, is made finite by the process of instantiation. In the example depicted in figure 6, the bus class $\mathcal{B}_{\mathrm{amp}}$ is instantiated and placed in the context of three components. Thus, the number of transitions is bounded by the number of links which interconnect the bus with the surrounding components. In figure 7, there is the resulting transition system which represents this bus instance.

More complex types of bus classes modeling both synchronous and asynchronous coordination models can be defined following the scenario presented above. Together with the possibility of instancing different bus classes in the context of one particular network, this example demonstrates the heterogeneity of the VCD coordination layer.

Applying the notion of architectural compatibility, we can check whether the system is locally deadlock free with respect to the interaction of a particular component with the bus instance it is connected to. In the example of the specification of ABP protocol, it can be checked, that the component $Sender$ and each of the $Receiver$ components is compatible with the bus instance $\mathtt{BUF}$.

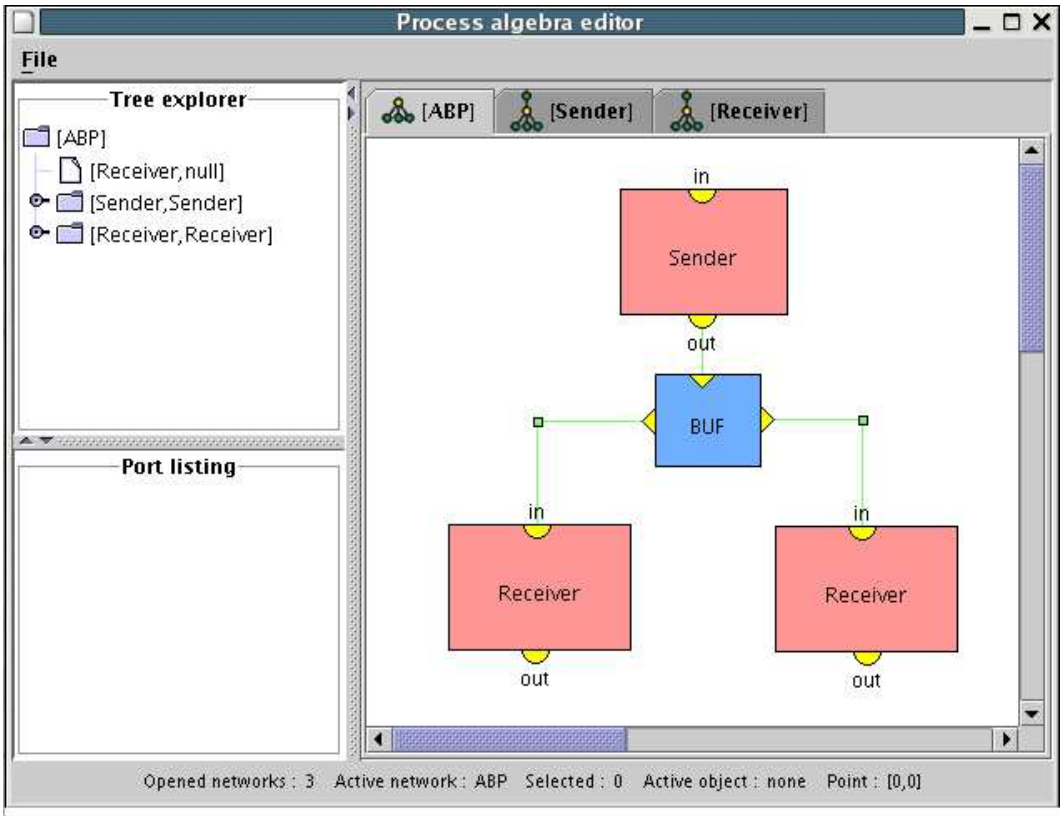

Figure 8: Main window of VCD editor

6 Tool support

We are currently implementing a prototype of an multiplatform editor for VCD diagrams. It is being implemented in Java. In fig. 8 there is a screenshot of the editor. The key features of the actual version of this editor are the following:

- Support for well arranged graphical specification of hierarchy of VCD networks. This is achieved by the notion of so called tree explorer, in which the hierarchy of networks can be easily viewed and managed.

- Support for checking of correctness of network dependencies at syntactic level. The editor contains a list of input and output ports of each network in the hierarchy and gives the user a help to bind the right subsystem ports to the higher ports in the network hierarchy.

- Reusing of already defined VCD networks. Together with architectural compatibility checking it will allow to easily define new VCD models by composing components which were already defined.

The actual version of the software is only a prototype which supports editing of the VCD syntax together with checking of syntactic correctness. We are currently working on implementation of the semantics defined in this paper.

7 Conclusions and Future Work

In this paper we have presented the formalism VCD for hierarchical specification of heterogeneous system architectures. The key concept of the language are buses which represent coordination models used in system architectures.

We see the main contributions of our work in three ways. First of all, the component-based character of VCD together with its hierarchical structure based on precise operational semantics allows to join the traditional design methods with the formal methods known from the theory of process algebras (e.g., refinement, equivalence or model checking [5]). On the other hand, the both syntactical and semantical separation of modeling the coordination aspects from modeling the behavioral aspects makes it possible to define a static communication infrastructure of a system independently of modeling the behavioral parts (the exogenous model [3]). Finally, heterogeneity supported in both behavioral and coordination layers of the language allows not only mixing of various coordination models in one specification, but also using of different models for behavioral description of components. For example, it is possible to put components defined as Statecharts together with components defined as Petri Nets and specify coordination relations among them using the constructs of the VCD coordination layer.

In our future work, we would like to add the typed value-passing support to VCD. We also aim to make a precise analysis of relations of our language with other formalisms, especially with process algebras. We would like to bring the notion of equivalences known from process algebraic theories and adapt them to VCD. In the future work on tool support, we aim to connect the editor of VCD with the distributed verification environment DiVinE [15].

References

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[2] D. Antoš, O. Fučík, and J. Novotný. Project of IPv6 Router with FPGA Hardware Accelerator. In *Proceeding of 13th International Conference on Field-Programmable Logic and Applications*, volume 2778, pages 964–967. LNCS, Springer-Verlag, 2003.

[3] P. Ciancarini. Coordination Models and Languages as Software Integrators. *ACM Computing Surveys*, 28(2):300, 1996.

[4] R. Cleaveland, X. Du, and S. A. Smolka. GCCS: A Graphical Coordination Language for System Specification. In *Proceedings of COORD'00*. LNCS, Springer Verlag, 2000.

[5] Orna Grumberg Edmund M. Clarke and Doron A. Peled. *Model Checking*. Cambridge : MIT Press, 1999.

[6] D. Harel. Statecharts: A Visual Formalism for Complex Systems. Technical report, The Weizmann Institute, 1987.

[7] D. Harel and H. Kugler. The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML). In *Proc. of 3rd Int. Workshop on Integration of Software Specification Techniques for Applications in Engineering*, volume 3147, pages 325–354. LNCS, Springer-Verlag, 2004.

[8] D. Harel and B. Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff. Technical Report MSC00-16, Weizman Institute of Science, 2000.

[9] J. Holeček Jan, T. Kratochvíla, V. Řehák, D. Šafránek, and P. Šimecek. How to Formalize FPGA Hardware Design. Technical Report 4/2004, CESNET z.s.p.o., 2004.

[10] S. Leue. *Methods and Semantics for Telecommunications Systems Engineering*. PhD thesis, University of Berne, 1994.

[11] J. Magee and J. Kramer. Dynamic Structure in Software Architectures. *SIGSOFT Softw. Eng. Notes*, 21(6):3–14, 1996.

[12] A. Maggiolo-Schettini, A. Peron, and S. Tini. A Comparison of Statecharts Step Semantics. *Theoretical Computer Science*, 290, 2003.

[13] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[14] OMG. *Unified Modeling Language. Version 2.0*. OMG, 2003.

[15] ParaDiSe Lab, Masaryk University Brno. *DiVinE project home page*, 2004.

[16] F. Plášil, D. Bálek, and R. Janeček. SOFA/DCUP: Architecture for Component Trading and Dynamic Updating. In *Proceedings of the International Conference on Configurable Distributed Systems*, page 43. IEEE Computer Society, 1998.

[17] A. Ray and R. Cleaveland. Architectural Interaction Diagrams: AIDs for System Modeling. In *Proc. of ICSE 2003*. IEEE, 2003.

[18] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Trans. Softw. Eng.*, 21(4):314–335, 1995.

[19] Michael von der Beeck. Formalization of UML-Statecharts. In *Proceedings of UML 2001*, LNCS. Springer-Verlag, 2001.

[20] D. Šafránek. SGCCS: A Graphical Language for Real-Time Coordination. In *Proceedings of FOCLASA'02*, volume 68 of *ENTCS*. Elsevier Science, 2002.