Visual Coordination Diagrams *

David Šafránek

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
xsafran1@fi.muni.cz

Abstract. In this paper, the work on a visual specification language for high-level design of concurrent systems with heterogeneous coordination models, called Visual Coordination Diagrams (VCD), is presented. Special emphasis is given to component-based specification of heterogeneous systems. The key property of VCD is separation of behavioral aspects from coordination aspects.
We briefly summarize recent research related to the topic and give an overview of VCD language and its semantics. The summary of already achieved results of our work is presented as well.

1 Research question

For specification and design of software systems visual software engineering methods are very popular [2]. The most wide-spread methods are included in the Unified Modeling Language (UML). They appear also very useful for the specification of a wide spectrum of concurrent systems.

Concurrent systems can be taken as groups of components running in parallel and interacting together via some communication media. With respect to this fact, concerning formal specification, we distinguish *behavioral* and *coordination layer* of concurrent systems. An ideal visual formalism for specification of concurrent systems should be capable of handling both the coordination and the behavioral aspects of concurrent systems. In general, there are two groups of visual formalisms for concurrent systems. The first group is formed by *state-based formalisms* (e.g., Statecharts), the second one contains *data-flow-based formalisms* (e.g., Message Sequence Charts). Both approaches emphasize different aspects of modeled systems and can be combined in a particular design.

The importance of universal modeling languages such as UML is very significant in software engineering. The required properties of such an universal design language, which would be suitable for concurrent systems, are its heterogeneity, hierarchy and component-based structure. Nowadays, typical computerized systems are composed from hardware and software components with different models of computation, some of them may be transformational, while the others can be interactive or reactive. We call such complex systems *heterogeneous* [4].

* This work has been supported by the Grant Agency of Czech Republic grant No. 201/03/0509 and FRVS grant No. 504/2004.

Unfortunately, there are no *formal* visual languages with appropriate universal character. Todays research on specification of systems is aimed at formalization of structures exhibited in UML. Adaptation of these ideas (especially collaboration diagrams and state-control diagrams) is also the topic of our research.

1.1 Objectives of our research

The main objective of our research is to define an universal formal visual language for concurrent systems, incorporating both the state-based and the dataflow based approaches in one formalism. Moreover, inspired by the work of [3],[4] and following our previous work on this topic, we would like to separate these two approaches into two independent layers of the language. We would like to achieve heterogeneity at both layers. Especially, we are primarily focused on the coordination layer. The main idea of our approach is to obtain an universal visual language for specification of synchronous and asynchronous coordination of heterogeneous components, which have their behavior specified in possibly different semantically compatible formalisms. The most significant properties we are taking into account are the compositionality and the hierarchy of the coordination layer, which is an important factor in component-based design.

2 Current solutions

In the field of state-based visual formalisms for concurrent systems, 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 hyper-graphs, the hierarchy of states was added, leading to Harel's Statecharts [5]. The richness of Statecharts syntax has shown that reaching a compositional formal semantics for such a powerful language is tedious. A various sub-dialects of Statecharts have been defined to achieve required semantic properties [8]. The concept of Statecharts was also incorporated into UML and formalized in [11].

In dataflow-based formalisms the concept of message flow graphs is employed to visually describe partial message passing interaction between concurrent processes. The high level message flow diagrams called Message Sequence Charts are based on this concept [6]. This formalism does not support hierarchical design. For its simple nature, it is widely used in telecommunication industry and it is also a part of UML.

Graphical calculus of communicating systems (GCCS) [3] adapts the robust process algebraic approach as its formal underlying semantic model. In contrast to state-based formalisms extended with concurrency and communication, in GCCS there are coordination aspects strictly semantically separated from behavioral aspects. The language has component-based hierarchical architecture. However, GCCS is too tightly related with process algebraic binary handshake communication. It has been proved to be inconvenient to express some non-trivial coordination mechanisms like broadcast using this process algebraic primitive [9].

Architectural Interaction Diagrams (AID) [10] are similarly as our VCD based on the idea of GCCS. VCD and AID both achieve some level of heterogeneity by breaking the relation with CCS process algebra. One of the significant differences between these two formalisms is in the underlying semantic model. AID is oriented to the 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 then AID, and thus allows for more heterogeneity at this level. On the other hand, AID allows more non-deterministic modeling at the coordination layer.

3 Preliminary ideas and the proposed approach

The proposed language, Visual Coordination Diagrams (VCD), focuses on the coordination level. The behavioral level is represented by state transition diagrams. Each *component* of the system has its own behavioral description, which is independent of other components in the system. At the coordination layer, components are grouped into *networks*. Any component together with its *interface*, containing input and output *ports*, can be embedded into a network. Interfaces are interconnected using special components with predefined behavior – so called *buses*. Buses 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 mechanisms can be effectively specified using a single uniform formalism.

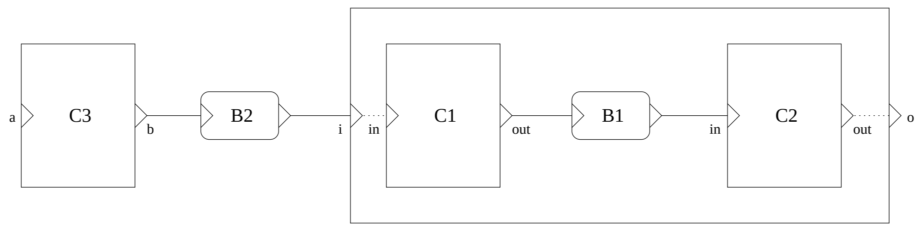

Fig. 1. Network and its hierarchy

The key concept of VCD is in its hierarchical network structure. This is achieved by the feature of taking networks as components of other networks (higher level networks). An example of network nesting is given in Fig. 1.

An example of a more complex network is in Fig. 2. It is an abstract simplification of a design specification of a hardware accelerated network router [1]. This example is interesting to be presented here because it highlights some features of VCD. The network in the figure describes the basic components of the router hardware and connections among them. The role of the whole system is to take input network packets on its hardware input (port *hw_in*), reading the routing and firewalling tables on its software input (port *sw_in*) and produce the packets modified according to these tables on the *hw_out* output port. Additionally, the system manages and makes some statistics about packets which flow through it. To achieve this functionality, it takes some software configuration on its *pm_config* port and produces the relevant software output on the *sw_out* port.

Refining the system into components, there are four different units interconnected by two buses. The unit *PPU* (Packet Parsing Unit) identifies the information important for routing and firewalling from input packets. The extracted information is synchronously sent to three other units. Note that the synchronous communication is modeled by the broadcast bus *BC*. In the *PEU* (Packet Editing Unit) unit the sent information is stored in some kind of memory for later use. In the *PM* (Packet Manager) unit it is used for computing some statistics. Finally, in the *RFU* (Routing and Firewalling Unit) it is compared with information in the tables, modified and sent asynchronously through the *BUF* bus to the *PEU* unit.

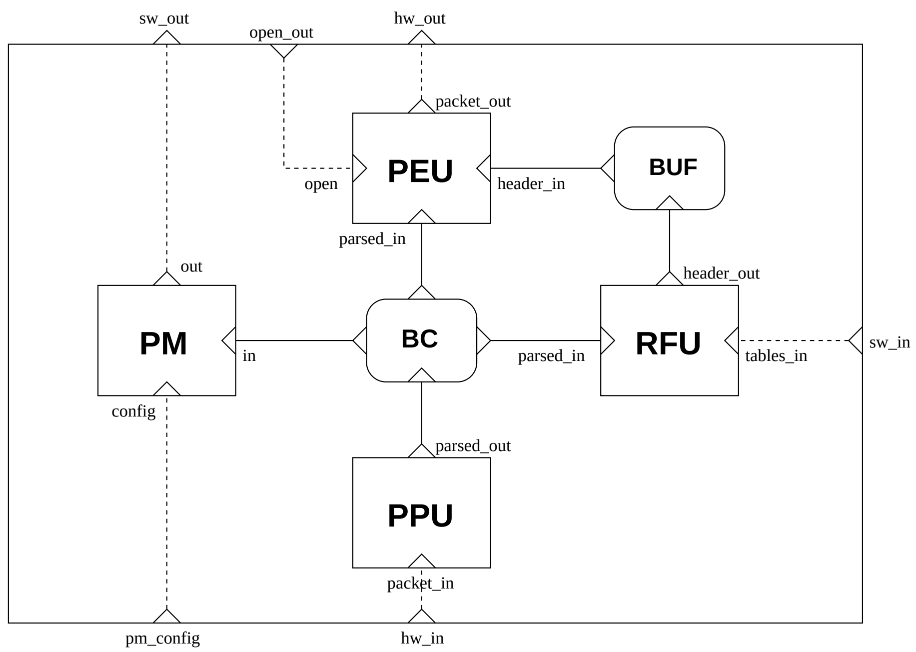

Fig. 2. VCD diagram of a routing and firewalling system

3.1 Coordination layer

Semantics of VCD is based on state transition model. 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 achieved using the Plotkin-style inference rules with respect to the VCD network structure. In this section we briefly sketch the semantics of basic VCD constructs. The precise formal semantics with the inference rules is presented in [14].

Ports and Interfaces The most basic element of the coordination layer is a *port*. We fix $\mathcal{W}$ a countable set of *write port names* and $\mathcal{R}$ a countable set of *read port names*. We require $\mathcal{W} \cap \mathcal{R} = \emptyset$. *Interface* I is then defined as any non-empty set of port names $I \subseteq \mathcal{W} \cup \mathcal{R}$.

Buses and Bus Classes The key construct of the coordination layer is a *bus*. As mentioned above, buses represent coordination mechanisms. They are used for modeling of various types of coordination models, such as bi-party handshake message passing, synchronized broadcast, or asynchronous types of component coordination. Different buses can be mixed in the specification of a particular network, which makes the coordination layer heterogeneous. Individual types of coordination mechanisms are represented as *bus classes*, which are formally defined as input/output labeled state transition systems (I/O LTS). See the following definition.

Definition 1. Bus class $\mathcal{B}$ *is a tuple* $\langle Q, T, q_0 \rangle$ *where*

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

Note that as buses behave inversely to components (compare with the definition presented in the following subsection), input port names of a bus correspond to port names from the set $\mathcal{W}$.

Any bus class can be instantiated as a particular bus and used for specification of coordination of components in a network. The bus interface is always given by the set of links which connect the bus to the ports of surrounding components. Formal definition of a bus, given by its interface and its class, is the following.

Definition 2. Bus B *of a bus class* $\mathcal{B}$ *is a tuple* $B = \langle I, \mathcal{B} \rangle$*, where* I *is an interface and* $\mathcal{B}$ *a bus class.*

As an example of a bus class definition, we give here the coordination model of synchronous multicast interaction. Its behavior is to non-deterministically choose an event involved on one of its input ports, replicate that event and synchronously transfer it to all its output ports.

Formally we define class $\mathcal{B}_{mc}$ of synchronous multicast buses as the following one-state I/O LTS:

$$\mathcal{B}_{mc} = \langle \{q_0\}, T, q_0 \rangle$$

Transition relation T is infinite countable set defined by the following expression:

$$\forall w \in \mathcal{W}, \Delta \subseteq \mathcal{R}, \Delta \neq \emptyset . \langle q_0, \{w\}, \Delta, q_0 \rangle \in T$$

In the example of the BC bus from the figure 2, the BC bus is an instance with one input and three output ports. Note that the process of instantiation of a bus class puts a finite bound on the state transition space of a bus.

3.2 Behavioral layer

We define the semantics of the component behavior as a Mealy machine, formally a tuple $\langle Q, R, q_0 \rangle$ with

- Q set of states,
- q_0 the initial state,
- $R \subseteq Q \times 2^{\mathcal{R}} \times 2^{\mathcal{W}} \times Q$ transition relation.

We use the similar synchronous/reactive principle as in Statecharts [7] for triggering a transition. Note that here we have simplified the Statecharts triggering model in the following way. All the input events in the input set of a particular transition are taken conjunctively. We do not allow to require absence of an event, so in this sense our model is weaker then the Statechart model. In Fig. 3 there are simple Statecharts which model behavior of some components of the network of Fig. 2.

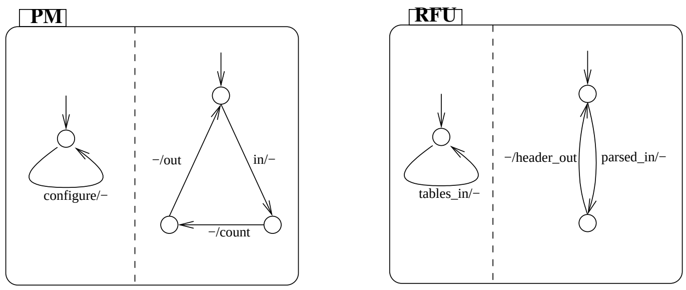

Fig. 3. Statecharts modeling behavior of components

4 Contributions and results of the work

In [12], we have elaborated the idea of a coordination framework [3] of a visual language GCCS for interactive systems based on asynchronous process-algebraic semantics of the CCS calculi. For description of behavioral layer, we have used finite Moore machines. Moreover, we have implemented an editor for the language, which supports visual specification of the coordination layer, while components are behaviorally specified in CCS. According to the process algebraic semantics of GCCS, the editor supports transformation of the entire GCCS specification into CCS in syntax of the Concurrency Workbench of New Century verification tool developed at Stony Brook, State University of New York. There is no value-passing support in the language.

In [13] we extended GCCS to its synchronous version. The process-algebraic semantics of this extension is based on SCCS. The semantics of the language is defined in terms of SCCS expressions. The concept of buses is refined in the way we sketched in the previous section. More precisely, buses have defined their state transition behavior as stand-alone components. However, there is no value-passing support in the language. Despite this limitations, the tight connection with the SCCS algebra makes this language useful for visual specification of reactive systems which can be directly verified using the concurrency workbench tool.

4.1 Future work

To achieve the heterogeneity at the behavioral level of our language, we would like to incorporate various state-based formalisms to serve for specification of components. We are going to enhance the behavioral layer with some suitable subset of Statecharts. In connection with adding such an expressive state-based formalism, the question of compositionality of the behavioral layer arises. However, the compositionality of the coordination layer is the basic assumption for our language. In consequence of adding state hierarchy into the behavioral layer, we will discuss, if possible support for local concurrency does not decrease the language comprehensibility. Further we would like to add some subset of Petri nets into our language.

Concerning the implementation, we are currently implementing a graphical tool which allows VCD diagrams to be simply created and modified. We develop it in Java. It is intended to be a tool for visual specification of real systems as well as a framework for easy implementation of future extensions of the language. The compositionality of the language allows us to equip the editor with the support of component-based design methods, such as top-down design, reusing of components, etc. We want to relate the editor with suitable formal verification tools.

We would like to use our language together with the implemented editor in practical design. Especially, we plan to experimentally use our language for description of hardware circuits and process control systems as well as for description of communication protocols and distributed software.

References

1. J. Barnat, T. Brázdil, P. Krčál, V. Řehák, and D. Šafránek. Model checking in IPv6 Hardware Router Design. Technical Report 07, CESNET, July 2002.
2. Dines Bjoerner. New results and trends in formal techniques for the development of software for transportation systems. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. Techn.Univ. of Braunschweig, 2003.
3. 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.
4. A. Girault, B. Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design*, 18, June 1999.
5. David Harel. Statecharts: A Visual Formalism for Complex Systems. Technical report, The Weizmann Institute, 1987.
6. Stefan Leue. *Methods and Semantics for Telecommunications Systems Engineering*. PhD thesis, University of Berne, 1994.
7. Gerald Lüttgen, Michael von der Beeck, and Rance Cleaveland. Statecharts via Process Algebra. Technical report, Langley Research Center, 1999.
8. A. Maggiolo-Schettini, A. Peron, and S. Tini. A comparison of statecharts step semantics. *Theoretical Computer Science*, 290, 2003.
9. K. V. S. Prasad. Broadcast Calculus Interpreted in CCS upto Bisimulation. In *Proceedings of EXPRESS'01*. ENTCS, 2001.
10. Arnab Ray and Rance Cleaveland. Architectural Interaction Diagrams: AIDs for System Modeling. In *Proc. of the 25th International Conference on Software Engineering (ICSE 2002)*. IEEE, 2003.
11. Michael von der Beeck. Formalization of UML-Statecharts. In *Proceedings of UML 2001*, LNCS. Springer-Verlag, 2001.
12. David Šafránek. Graphical Specification of Concurrent Systems (in Czech). Master's thesis, Faculty of Informatics, Masaryk University Brno, 2001.
13. David Šafránek. SGCCS: A Graphical Language for Real-Time Coordination. In *Proceedings of FOCLASA'02*, volume 68 of *ENTCS*. Elsevier Science, 2002.
14. David Šafránek. Visual Specification of Systems with Heterogeneous Coordination Models. Submitted. http://www.fi.muni.cz/~xsafran1/publications/foclasa04entcs.ps.gz, 2004.