Visual Specification of Concurrent Systems

David Šafránek
Department of Theory of Programming
Faculty of Informatics
Masaryk University Brno
xsafran1@fi.muni.cz

Abstract

The work on a visual formalism for specification of concurrent systems is presented. It is proposed to match requirements of state-of-the-art component-based design methods. Special emphasis is given to specification of heterogeneous systems in which the different models of computation can be mixed together.

We briefly summarize recent research related to the topic and give a sketch of the basic ideas for definition of the proposed language. The already achieved results of our work are presented as well.

1 Research question

For specification and design of software systems visual software engineering methods are very popular. The most wide-spread methods are included in the Unified Modelling Language (UML). They appear also very useful for the specification of a wide spectrum of concurrent systems. Development of formal visual languages for modelling of systems allows application of formal methods in early phases of system design. Visual languages have the advantage of being simple to use for system designers. A difficult problem is to find a compromise between the richness of syntactic constructs and the comprehensible formal semantics of visual languages. Ideally, a visual language should be able to handle both the coordination and the behavioral aspects of concurrent systems. In general, there are two groups of visual languages for concurrent systems. The first group is formed by *state-based languages*, the second one contains *data-flow-based languages*. Both approaches emphasize different aspects of modelled systems and can be combined. Another well-known visual formalism are Petri nets. Due to the fact that both coordination and behavioral aspects are inseparably modelled with the notion of a token, this formalism is not useful for visual modelling of complex systems.

The importance of universal modelling 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*. 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 one of the topics of our research.

1.1 Objectives of our research

The main objective of our research is to propose, develop and implement an universal formal visual language for concurrent systems, incorporating both the state-based and the data-flow based approaches into one formalism. Moreover, inspired by the work of [1],[2] and following our previous work on the topic, we would like to separate these two approaches into two independent levels of the language in the sense of syntactic and semantic independence. We would like to achieve heterogeneity at both levels. Especially, we are primarily focused on the coordination level. 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 level, which is an important factor in component-based design.

2 Current solutions

In the field of state-based visual languages 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 [3]. The complexity of the syntactic richness of Statecharts 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 [7]. The concept of Statecharts was also incorporated into UML [8].

In dataflow-based languages 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 [5]. 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.

For complex concurrent systems it is useful to separate the behavioral (state-transition) parts from the communication (data-flow) parts in the specification. As this is the main motivation for our work, we briefly mention two recent solutions related to this topic.

Graphical calculus of communicating systems (GCCS) [1] adapts the robust process algebraic approach as its formal underlying semantic model. In contrast to state-based languages extended with concurrency and communication, the coordination aspects are strictly semantically separated from behavioral aspects. The language has component-based hierarchical architecture. However, GCCS is a framework and is too tightly related to the handshake type of communication in CCS. Hence, its heterogeneity is low.

Another approach, presented as *charts [2], is aimed to establish strong heterogeneity. The language is based on hierarchical structure, in which behavioral aspects can be mixed with the data-flow aspects at an arbitrary level of abstraction. In contrast to GCCS, the states of the behavioral parts can be refined with data-flow diagrams and vice-versa going deeper in the hierarchy. Diversity of different computational models mixed together makes it difficult to define formal operational semantics. Due to its character, the language is not a practical design language.

3 Preliminary ideas and the proposed approach

The proposed approach is inspired by GCCS. Two-level language separating the specification of behavioral aspects from the coordination aspects is defined. The behavioral level is represented by state-transition diagrams. Each *component* of the system has its own behavioral description, which is independent on other components in the system. The coordination level combines components into *networks*. Each component together with its *interface* containing input and output *ports* can be embedded into a network. Interfaces are interconnected by a special components with predefined behavior – so called *buses*. Buses specify the interaction among the components in the network. The coordination level is hierarchical. A network with its interface can be embedded as a component into a higher-level network.

An example of a network with four low-level components representing the behavioral blocks of a simple traffic light system is given in figure 1. The interface of the entire network has only one port labeled *push*, it is the input port. The environment of the network (a pedestrian) can interact with the system by sending a signal to this port.

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

- Q finite set of states,
- q_0 the initial state,
- Σ_I, Σ_O finite sets of input and output signals,
- $R \subseteq Q \times 2^{\Sigma_I} \times 2^{\Sigma_O} \times Q$ transition relation.

We use the similar synchronous/reactive principle as in Statecharts [6] for enabling a transition. In our example, if the *push* signal is present and the component *ctrl* is in its initial state, the reaction (*go*) is emitted instantaneously and the component moves to another state. The signal *push* is removed ("eaten") from the environment.

The components in the figure are interconnected with buses *B1* and *B2*. We have chosen a synchronous model of interaction and we would like these buses to serve as synchronous replicators of incoming signals to all their outputs. In other words, we model instantaneous broad-cast.

In general, we would like to support various models of interaction (different types of buses). To handle this, we define the semantics of a bus as a Mealy machine with transition labels extended by the information about the actions happening on the ports of components connected to the bus. Formally, the bus is a tuple $\langle Q, R, \Sigma_I, \Sigma_O, \Sigma_{EI}, \Sigma_{EO}, q_0 \rangle$ where

- $\Sigma_{EI} \subseteq \Sigma_O$, $\Sigma_{EO} \subseteq \Sigma_I$ denote enabled actions in the input and output ports of the interfaces of components connected to the bus respectively,
- $R \subseteq Q \times 2^{\Sigma_{EO}} \times 2^{\Sigma_I} \times 2^{\Sigma_{EI}} \times 2^{\Sigma_O} \times Q$.

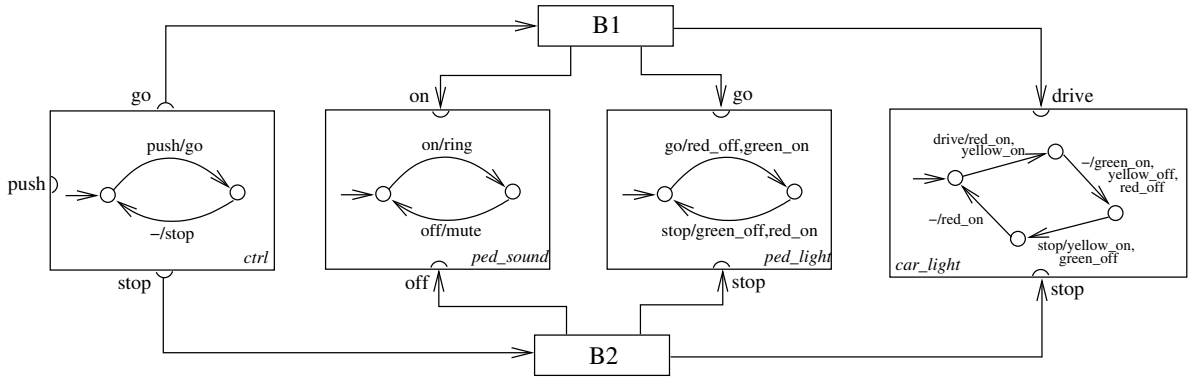

Figure 1. Example of a network

The other components of the tuple are the same as above. As we sketch the semantics only briefly here, we have to note that the mapping of component actions to corresponding bus actions which match the EI and EO sets is done according to the links among components and a bus.

The bus for instantaneous broad-cast can be defined in general manner as follows:

- $Q := \{q_0\}$ contains a single state.
- Σ_I, Σ_O contain all the actions that can happen in a network.
- R contains all transitions satisfying:
  - $EO \neq \emptyset, EI \neq \emptyset$
  - $\exists a \in EO.\ \langle q_0, \{a\}, \{a\}, EI, EI, q_0\rangle \in R$

Intuitively, a bus transition is fired whenever there is a signal sent to the bus. In the same instant of time, each component which is connected to the bus by an input port and has enabled (included in EI) transition triggered by the signal on this port receive the broad-casted signal. E.g., assume all the components in the figure are in their initial states and $B1$ is a broad-cast bus defined by the definition above. Then $EO := \{go\}$ and $EI := \{on, go, drive\}$. The bus transition $\langle q_0, \{go\}, \{go\}, \{on, go, drive\}, \{on, go, drive\}, q_0\rangle$ can be fired modelling the instantaneous broad-cast of the signal go to the components ped_sound, ped_light and car_light.

The operational semantics of the network can be defined by Plotkin-style inference rules, which combine transitions of the components (including buses) into a network transition. The invisible τ-signal is introduced with the similar meaning as the τ-action in the CCS calculus. In our example, the $ring$ signal of the *ped_sound* component appears as τ at the network level. It is because the interface of the component does not include the port $ring$ (a case of abstraction).

Our approach allows specification of networks with different types of buses. We can define buses for asynchronous and synchronous communication and mix them in a particular network. Moreover, we can use an arbitrary state-based formalism which is semantically compatible with Mealy machines (e.g. Statecharts, Argos, SyncCharts, etc...) to describe the behavior of a component. Components specified by different formalisms can be contained in the same network. This is how we achieve heterogeneity of the coordination level as well as the behavioral level.

4 Contributions and results of the work

In [9], we have elaborated the idea of a coordination framework [1] of a visual language GCCS for interactive systems based on asynchronous process-algebraic semantics of the CCS calculi. For description of behavioral level, we have used finite Moore machines into GCCS. Considering succinctness, this approach is comparable to communicating finite state machines, with addition of hierarchy into the coordination level. Moreover, we have implemented an editor for the language, which supports visual specification of the coordination level, while the 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 [10] we extended GCCS to its synchronous version. The process-algebraic semantics of this extension is based on SCCS. The coordination level is equipped with both asynchronous and synchronous semantics. 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

represented formally as SCCS agents. 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.

In [11] we have refined the concept of buses to fit the needs of hardware and process-control design. Buses are extended with internal ports and support broad-cast and multi-cast synchronous communication. A framework for more complicated signal distribution is captured by the notion of the so-called resolution buses.

4.1 Future work

In our future work, we are going to follow the idea [2] of developing the universal coordination framework for different models of computation. We aim to adapt the underlying process algebraic semantics to be more feasible for the needs of the universal coordination framework. Also, we are going to equip our language with the typed-value-passing support. In consequence, we would like to rigorously compare the succinctness and semantic expressiveness of the coordination level of the developed language with other languages.

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 level with some suitable subset of Statecharts. In connection with adding such an expressive state-based formalism, the question of compositionality of the behavioral level arises. However, the compositionality of the coordination level is the basic assumption for our language. In consequence of adding state hierarchy into the behavioral level, 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. The expressiveness of such extensions will be rigorously evaluated. In the case of hierarchical state-based behavioral model, we will have to compare expressiveness of our language with Statecharts itself. In the case of incorporating of Petri nets, the comparison with the box algebra [4] will be done.

Considering the implementation, we are going to design and implement the user interface for the developed language with respect to its universality. The editor of the language is intended to be a practical tool for visual specification of complex 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 apply our language together with the implemented interface 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] 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.

[2] A. Girault, B. Lee, and E. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design*, 18, June 1999.

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

[4] M. Koutny and E. Best. Operational and denotational semantics for the box algebra. *Theoretical Computer Science*, 211, 1999.

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

[6] G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via Process Algebra. Technical report, Langley Research Center, 1999.

[7] A. Maggiolo-Schettini, A. Peron, and S. Tini. A comparison of statecharts step semantics. *Theoretical Computer Science*, 290, 2003.

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

[9] D. Šafránek. Graphical Specification of Concurrent Systems (in Czech). Master's thesis, Faculty of Informatics, Masaryk University Brno, 2001.

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

[11] D. Šafránek. A Graphical Synchronous Language GSCCS. Technical report, FI MU Brno, 2003. http://www.fi.muni.cz/∼xsafran1/publications/gsccs.ps.gz.