SGCCS: A Graphical Language for Real-Time Systems ⋆

David Šafránek

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

Abstract. We present a graphical language SGCCS as a language for modeling of discrete real-time systems. SGCCS can be viewed as a graphical version of the synchronous Calculus of Communicating Systems (SCCS). A specification in SGCCS contains both graphical and visual components. We give an overview of basic concepts of SGCCS giving an example of specification in SGCCS. Further, we show briefly how the formal semantics of SGCCS is developed.

1 Introduction

In [2], a graphical language for specification of interaction among concurrent heterogeneous components GCCS was presented. We implemented a simple editor for this language ([4]) with support for translation of a graphical model into the Calculus of Communicating Systems (CCS [9]).

Due to our participation on a current project of a formal design of a hardware IPv6 router [7], we have been motivated to extend our graphical editor to support real-time design. It lead us to extend semantics of GCCS to deal with time. We adapted the notion of the synchronous extension of CCS (SCCS, [9]). The result is called SGCCS.

In this paper, we would like to present SGCCS briefly as a graphical formalism for formal visual component-based modeling of systems, for which time plays critical role. We assume time being discrete and global. All components are observed as performing their actions in time-slots. Any time-slot can be seen as a list of actions to be fired simultaneously at a discrete tick of the global clock.

It is worth noting that there are two different notions of synchrony in our language. First one is the concept of synchronous communication of components in the sense of instantaneous handshake or multicast interaction among them, and the second one is the concept of time-slots we have mentioned above.

2 Related work

There exist some graphical formalisms for modeling of real-time systems. One of them is the language GCSR [1], which is based on the idea of executing inter-

⋆ This work has been partially supported by the Grant Agency of Czech Republic grant No. 201/00/1023.

leaved actions asynchronously. There is also a large group of graphical formalisms based on the notion of Statecharts, e.g., its extension Timed Statecharts [8]. Another well-known graphical formalism are Petri-Nets and their extensions. In general, in contrast to SGCCS, we cannot easily distinguish between the interaction and the behavioral level of specification in these formalisms. Interaction aspects are closely related with behavioral aspects and cannot be simply separated. In other words, these formalisms cannot be simply used as an interaction "umbrella" for components with their behavior specified in different formalisms.

3 Overview of Synchronous GCCS

The action structure of SGCCS is the same as in Synchronous CCS. The notion of *particle* and *composite actions* is used. Particle actions are the smallest distinguishable action elements. By composing particles using the operation of product we get composite actions. In a particular time-slot, the system can perform a composite action. Thus, particle actions are not atomic in the sense of being performed exclusively, they are fired simultaneously with other particles which together form the complete composite action performed in the current time-slot. We denote the composite action α containing the particles a, b, c using the dot-notation: $\alpha = a.b.c.$

As in GCCS, systems in SGCCS are graphically specified at two levels – the *process* level and the *network level* (the *hierarchy* of networks). At the process level, the behavior of components is specified using transition diagrams with edges labeled with input and output actions. An example of a process level specification is showed in the right part of Fig. 1.

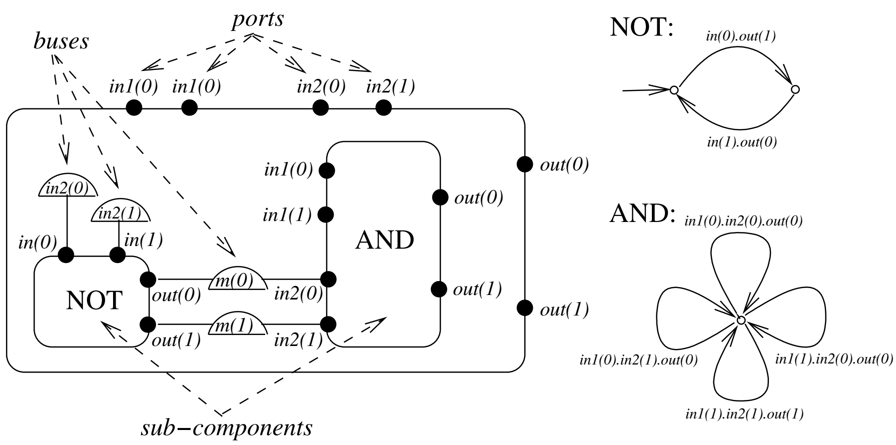

Fig. 1. The network and the process level of SGCCS

Interaction among components is specified at the network level using so called *nets*. An example of a net is in the left part of Fig. 1. Components are included in a net as boxes, which represent *interfaces* with places for communication – so called *ports* (black points in Fig. 1). Each port is labeled with the name of some action exporting it to the environment of the component. We consider ports as bi-directional, what means that both input and output actions can flow through them. The synchronous model allows to signal arbitrarily many actions simultaneously in a particular time-slot via ports of a particular component (e.g., $in1(0), in2(0)$ and $out(0)$ of the AND component in Fig. 1).

As is depicted in Fig. 1, components are connected by ports to so called *buses* (half-ellipses). Buses export actions involved on these ports to be observed by the environment outside of the net under the buses' names. Therefore, the actions $in(0)$ and $in(1)$ of the component NOT in Fig. 1 can be seen as $in1(0)$ and $in1(1)$ by the environment. Therefore there are ports related with these actions in the interface of the top-most net. The environment can be specified by another net, so called higher level net. The current net is embedded into the higher level net as one of its components. This creates the hierarchy of nets, which can be viewed as a tree. In the root of such a tree there is the top-most net. This tree structure goes with the modularity, refinement, and using of reusable components features in design using SGCCS.

4 Two models of synchronization

Buses can also model synchronization. By the term *synchronization* we mean handshake between two components. Due to the concept of synchronous firing of actions, all handshakes desired in a particular point of time are sensed to be performed in the same time-slot. In general, we can distinguish between two kinds of synchronization. Firstly, we can strictly require synchronization to take place in the current time-slot, we call this *non-delayed synchronization*. This is useful for example for modeling of logical circuits as can be seen in Fig. 1 (described later). Secondly, we can leave the components, which cannot synchronize in the current time-slot, to wait until synchronization will be possible. We call this kind of synchronization *delayed synchronization*. The example of using this model of interaction is shown in Fig. 2 (also described later in this section).

Synchronization is modeled in SGCCS using the construct of *synchronous buses* denoted by half-ellipses. An example of a system which applies the concept of non-delayed synchronization is in Fig. 1. Components NOT and AND, interconnected by buses $m(0)$ and $m(1)$, run synchronously in parallel. Whenever NOT performs $out(0)$ or $out(1)$ (i.e., it is in each time-slot), AND must be able to perform $in2(0)$ or $in2(1)$ in the same time-slot. Otherwise, both components would be deadlocked. System in Fig. 1 models behavior of the logical circuit having the function of the common *and*-operation taking one of its inputs inverted.

An example of delayed synchronization is in Fig. 2. It is a model of a two-cell queue. Important role has the 1-action in the diagram of the process CELL which stands for possible waiting before firing *in*. The right CELL can wait until the left one has a value on its *out* port. This behavior extends to the whole queue, because it is modeled by replication of the process CELL. The concept of delayed synchronization allows one to model asynchronous (buffered) communication in SGCCS. This feature is built in the network level of SGCCS and represented by the construct of *asynchronous buses*.

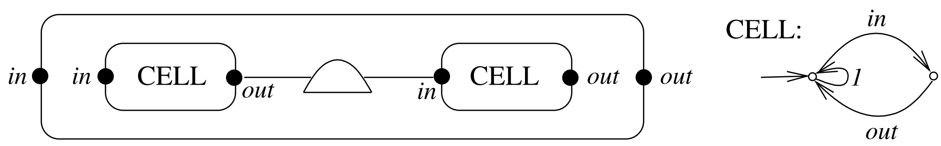

Fig. 2. An example of delayed synchronization

In general, arbitrarily many components can be connected by their ports to any bus. All possible handshakes are performed in one time-slot leading to an internal action. If two components can synchronize with two other components, then the non-deterministic choice of particular simultaneous synchronization pairs is applied. It is important to note, that the maximal set of all possible synchronization pairs is fired in one time-slot.

5 Developing the formal semantics of SGCCS

In [5], we presented formal definition of SGCCS. In general, a calculus of terms representing visual constructs of SGCCS is developed there. Semantics of these terms is defined via their mapping to SCCS expressions. Due to the lack of space, here we only show by example how the SCCS expression representing the semantics of the SGCCS net in Fig. 2 is derived. For general definition of SGCCS semantics we refer the reader to the cited paper.

First of all, we recall some basics of SCCS (see [9] for details). The parallel composition operator "$|$" known from CCS is replaced with the synchronous product operator "$\times$". We will also use the standard relabeling operator, and the restriction operator "$\backslash\backslash R$", where R is set of names of the restricted particles. The prefix operator ":" and the choice operator "+" will be sensed also in standard ways. The internal action is denoted as "*1*". Delayed δ-actions and the idling process **1** are defined by the following expressions:

$$\mathbf{1} \stackrel{df}{=} \mathit{1} : \mathbf{1}, \qquad \delta a \stackrel{df}{=} \mathit{1} : \delta a + a : \mathbf{1}$$

δ-action is correctly defined as the SCCS process which idles, then possibly performs the action, and after that idles forever.

The process layer of the buffer example in Fig. 2 is simple. It contains only one process CELL expressed in SCCS in the following manner:

$$\mathsf{CELL} \stackrel{df}{=} \delta in : \overline{out} : \mathsf{CELL}$$

We will express the net containing two copies of the CELL process as components using the product operator. The visible actions at the top-most level are those *in* and *out* actions, which belong to the free port *in* of the left CELL component, and *out* of the right CELL component. Ports connected to the bus are invisible. We will express the bus[1] as the synchronous product of two separate SCCS processes. The top-most net BUF is then expressed using the following definition:

$$\mathsf{BUF} \stackrel{df}{=} (\mathsf{CELL}[m/out] \times \mathsf{CELL}[m/in]) \backslash\backslash \{m\}$$

New added action m is used to connect the correct components. These actions are not observable at the level of the top-most net, i.e., they are restricted. In other words, the relabeling and restriction operators define the semantics of the synchronous bus.

Semantics of asynchronous buses is defined using a special SCCS process. The general case is rather technical, for that reason we give here only the simplified version. Imagine the bus in Fig. 2 is an asynchronous bus. For that case, the behavior of the bus is expressed by the following definition:

$$\mathsf{BUS} \stackrel{df}{=} \begin{array}{r} (\delta b_1 \times \bar{b}_2 : \mathsf{BUS}) + (\delta \bar{b}_1 \times b_2 : \mathsf{BUS}) \\ + (\delta b_2 \times \bar{b}_1 : \mathsf{BUS}) + (\delta \bar{b}_2 \times b_1 : \mathsf{BUS}) \end{array}$$

The top-most net BUF is then expressed using the following synchronous product:

$$\mathsf{BUF} \stackrel{df}{=} (\mathsf{CELL}[m_1/out] \times \mathsf{BUS}[m_1/b_1, m_2/b_2] \times \mathsf{CELL}[m_2/in]) \backslash\backslash \{m_1, m_2\}$$

Using this concept we also solve carefully possible name conflicts (i.e., ports and buses of the same name in the same network). The hierarchy of specification in our language is semantically based on the concept of abstraction in SCCS. Due to this property together with the independency of components, the reusability of component definitions is possible. This results into a language suitable for the modular component-based design.

6 Conclusion and future work

We presented a graphical formalism SGCCS for visual specification of discrete real-time systems. This language adds the notion of synchronous non-interleaved actions (so called composite actions) to the GCCS coordination language [2].

[1] To simplify the explanation, we considered the bus as the *unlabeled bus* here. In general, there are two sorts of buses considering labeling. Please, see the referred paper ([5]) for details.

Unlike the asynchronous GCCS, two different types of buses are distinguished in SGCCS, with respect to its synchrony. Using this two constructs one can model both delayed and non-delayed synchronization. We believe that this universality property could be useful for more complex systems which combine both software and hardware components.

The main advantage of SGCCS is that it is exogenous, which allows modeling of coordination of components without any detail knowledge about their behavior. This allows abstraction and application of the top-down methodology during the design phase. Concepts such this one are common in the component-based design [6].

We are currently working on the synchronous extension of the graphical editor [4]. Thus, analogously to the editor for GCCS, we aim to transform a model represented graphically in SGCCS into SCCS according to the definition of SGCCS semantics we have presented in [5]. Hence, one will be for example able to apply the μ-calculus model checking and equivalence checking to that model using the Concurrency Workbench tool [3].

Considering future work, we are going to add the value-passing feature to our language, i.e., we aim to develop a type system for messages that can be sent through particular ports and buses. Another future extension of SGCCS is to incorporate some other graphical formalisms to the process level of our language, i.e. Petri-Nets, which allow modeling of non-interleaving, and Statecharts, which has the feature of hierarchical modeling at the behavioral level. Other types of buses could be also added to our formalism to support instantaneous broadcast communication, message passing via finite buffers or any other coordination mechanisms which are possible to be modeled in SCCS.

References

1. H. Ben-Abdallah. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, 1997.
2. R. Cleaveland, X. Du, and S. A. Smolka. GCCS: A Graphical Coordination Language for System Specification. In *Proceedings of Fourth International Conference on Coordination Models and Languages*. LNCS, Springer Verlag, 2000.
3. R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In *Computer-Aided Verification (CAV '96)*, page 394. LNCS 1102, Springer-Verlag, 1996.
4. D. Šafránek. Graphical specification of concurrent systems (in czech). Master's thesis, Masaryk University, Brno, 2001.
5. David Šafránek. SGCCS: A Graphical Language for Real-Time Coordination. In *Proceedings of International Workshop on Foundations of Coordination Languages and Software Architectures*, volume 68.3. LNCS, 2002.
6. George T. Heineman and William T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison Wesley, 2001.
7. Jiři Barnat, Tomáš Brázdil, Pavel Krčál, Vojtěch Řehák, and David Šafránek. Model checking in IPv6 Hardware Router Design. Technical Report 08, CESNET, July 2002.
8. Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In *Formal Techniques in Real Time and Fault Tolerant Systems*. Springer-Verlag, 1991.
9. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.