Faculty of Informatics
Masaryk University

Graphical Specification of Concurrent Systems

David Šafránek

Ph.D. Thesis Proposal

Supervisor: Luboš Brim

Brno, February 28, 2003

Contents

Chapter 1

Introduction

The power of recent hardware technologies and the development of software methodologies enable to design very complex systems. As the complexity of systems in development raises, it is more and more difficult to ensure designed systems to be bug-free. Systems of high complexity are very error-prone. Consequences of an uncovered error could be very critical. It is extremely important to find all possible bugs and to find them as soon in the design process as possible.

Present modern computer-driven systems can be divided into the following three distinct classes:

transformational systems - This is a class of systems which compute the output values from its input values and stop. Most of numerical computation programs and compilers belong to this class of systems.

interactive systems - The basic property of systems from this class is constant interaction with the environment, in which they run. Additionally, this systems can be viewed as the leaders of the interaction. Whenever it can, the system listens to its environment, which calls for its services. The system delivers the services as soon as they are available. The typical interactive systems are operating systems, distributed databases, distributed algorithms, networking, etc. The main critical properties are deadlock avoidance, fairness, and coherence of distributed information.

reactive systems - This systems continuously react to stimuli coming from their environment by emitting back other stimuli. In contrary to interactive systems, reactive systems are purely input-driven and they must react in time dictated by the environment. In this systems, the environment is the leader of the interaction. Common members of this class are industrial process control systems, airplane or automobile control systems, embedded systems, audio and video protocols, hardware circuits, man-machine interfaces drivers, etc. The key critical properties are safety and timeliness.

Of course, every complex computer-driven system never falls entirely into any of these three classes. Nevertheless, it is always useful to identify which parts of the system are transformational, interactive or reactive, and to handle them with appropriate analysis and verification methods. The key property shared by both interactive and reactive systems is *concurrency*. At first, these systems act concurrently with their environment. At second, their internal parts are usually also concurrent and communicate with each other. For example the client and the server in the distributed database system or the time-keeper, stopwatch, and alarm in a digital watch.

Reactive systems differ from interactive systems in *determinism*. A system is said to be deterministic if the same sequence of inputs always produces the same sequence of outputs. It is nondeterministic otherwise. Reactive systems are deterministic. On the contrary, nondeterminism is the character of interactive systems. In general, nondeterminism is necessary for the specification and behavioral analysis of systems.

There exist several techniques for analysis, modelling, and proving the correctness of all the classes of systems mentioned above. These techniques, so-called *formal methods*, have a precisely formal mathematical base. The most important theory, which underlies formal methods, is the theory of formal semantics [Win93]. For proving the correctness of a system, its mathematical model must be given. Such a mathematical model is defined by the formal semantics of the system. In seventies, C.A.R. Hoare presented the approach called axiomatic semantics, for proving correctness of classical transformational systems written as imperative programs. In the last two decades, the main research topic on formal methods has been aimed to capture reactive systems, which are more complex and require different and more sophisticated approach. Formal analysis of such systems is based on their operational, denotational, and axiomatic semantics.

From this point onwards we focus on interactive and reactive systems only.

To analyze and prove critical properties of the system in design, formal description of the system and the specification of its properties have to be precisely done. Designers of the system have to choose an appropriate specification language to formalize the specification and the programming language for the implementation. This choice has to be done with respect to the fact which formal methods will be applied in various stages of design. E.g., in the case of using the *formal verification* method, one has to choose the appropriate *temporal logic* to apply the *model checking* method [EMCP99], or one has to choose other suitable specification language, which is closer to the *equivalence checking* [CS01] method.

Considering programming languages for implementation of systems, it is the fact, that classical software programming languages such as C/C++, Java or hardware design languages such as VHDL or Verilog are so complex that their complete clear formal semantics cannot be easily defined. Moreover, languages such as C/C++ or Java are not suitable for implementation of reactive systems. These were the reasons for foundation of languages which would have clear formal semantics and were suitable for reactive or interactive systems.

In the last two decades, several languages for both reactive and interactive systems were founded. In the field of interactive systems, languages CSP [Hoa85] and CCS [Mil89] based on the process algebraic approach have appeared. Their formal semantics and the theory behind them relate them naturally with formal verification methods. On these languages, which should be taken as design languages, the concurrent programming languages Ada and Occam were build. For reactive systems, process algebraic SCCS and Meije calculi became the base for Esterel [Ber98] language, around which a large group of so-called *synchronous languages* appeared.

Visual methods for specification of software with various design and analysis methods of software engineering, nowadays unified into the Universal Modelling Language UML, appear also useful for specification of reactive and interactive 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. In general, there are two groups of visual languages for concurrent systems. The first group is formed by *state-based languages*, which are based on ideas of Statecharts [Har87] (e.g., their synchronous variants Argos [Mar91], SyncCharts [APF00], or asynchronous GCSR [DS97],...). In the second group there are *data-flow-based languages*, build on ideas of Message Sequence Charts (e.g. UML scenarios and collaboration diagrams). Both approaches emphasize different aspects of modelled systems and can be combined. Another well-known visual formalism is Petri nets. Due to the fact that both coordination and behavioral aspects are inseparably modelled with the notion of a token, these formalism is not useful for visual modelling of complex systems.

Call for universal modelling languages such as UML is strong in present. The main advantage of such a language is its heterogeneity. Nowadays, computerized-systems are composed from components of different character, some of them maybe transformational, while the others can be interactive or reactive. We call such complex systems as *heterogeneous systems*. Unfortunately, there are no *formal* languages with appropriate universal character. Todays research on specification of systems is aimed at formalization of UML.

1.1 Objectives to be reached

The objective of this PhD work is to propose, develop and implement an universal formal visual language for concurrent systems, incorporating both the state-based and the data-flow based approach into one formalism. Moreover, inspired by the work of [CDS00],[GLL99] and our previous work on the topic, we would like to separate these approaches into two independent layers of the language in the sense of both syntactic and semantics independence. Especially, we are primarily focused on the communication level. The main idea of our approach is to serve as an universal visual language for specification of both 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.

Chapter 2

State of the art

In this chapter we briefly summarize visual formalisms for formal specification of reactive and interactive systems. We compare these formalisms and evaluate their properties.

We distinguish two different approaches for visual specification of reactive and interactive systems. These are the *state-based* and the *data-flow-based* approach.

In the *state-based* approach, the system is modelled as some kind of a state-machine. The behavioral aspects are explicit, on the other hand the communication aspects are implicit. State-based formalisms specify the internal behavior of communicating processes, while the communication behavior can be only indirectly inferred from the internal behavior.

In contrary, in the case of *data-flow-based* approach, the communication aspects are specified explicitly, while the behavioral aspects are implicit. Moreover, there is an abstraction from the behavioral aspects, because they are not specified in detail. Indeed, we take this approach rather as specification of the communication relations between processes than the specification of their behavior.

2.1 State-based methods

State-based formalisms are build upon the notion of *finite state machines*. The elementary object of a finite state machine is *state*. State can be taken as a function which assigns each variable of the system a value over its domain and the program counter. Each step of a computation is expressed as a *transition* from one state to another state. This model is suitable for both interactive and reactive systems. The finiteness of a state machine means that the set of all states is finite.

The computation of an interactive or reactive system relates to an infinite sequence of states and transitions. Transitions are instantaneous and discrete. While the system is in a state, the values of all variables remain unchanged. State machines for reactive systems differs from state machines for interactive systems in the way how they react with the environment and in determinism. State machines specifying reactive systems are assumed to be deterministic and synchronous with the environment. If the environment is modelled as another state machine then its composition with the state machine modelling the system is the synchronous product of both state machines. On the other hand, state machines modelling interactive systems are generally nondeterministic and their behavior is asynchronous with the environment. The composition of the state machine representing an interactive system and the state machine which models the environment is their asynchronous product. This asynchronous product models interleaving of transitions of both state machines.

Moore and Mealy machines

In the following definition the two basic types of finite state machines are defined. In general, we consider them as both nondeterministic or deterministic, depending on the character of the transition relation. The only difference between the *Moore* and the *Mealy machine* is that the Mealy machine has each of its transitions labelled with an input/output pair of symbols with the meaning of instantaneous firing of the output symbol whenever the environment gives the relevant input. On the other hand, the Moore machine can only accept an input symbol or emit the output symbol in each transition. Due to this fact, Moore machines are taken as the basic state-based formalism for interactive systems and deterministic Mealy machines for reactive systems. Graphically they are represented as state-transition diagrams. An example of a Moore and a Mealy machine is in figure 2.1.

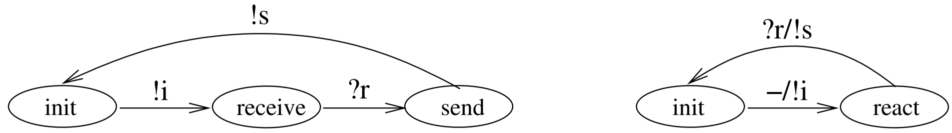

Figure 2.1: Moore and Mealy machines

These formalisms are formally defined in [HU79] and it is proved there that the Moore and the Mealy machines are expressively equivalent formalisms. While these formalisms form an important concept in behaviororiented specification, their simplicity implies their low succinctness. They have the following limitations:

1. missing data abstraction
2. missing abstraction and refinement of states and transitions
3. combinatorial exponential state space explosion due to concurrent composition
4. missing composition and synchronization mechanisms

The limitations mentioned above implies that these state machines are not useful for specification of complex systems directly. In the following subsections we give an overview of extensions of both formalisms which overcome some of these limitations.

Communicating finite state machines

Communicating finite state machines CFSMs is a formalism which overcomes the last two limitations of Moore and Mealy machines. CFSM are composed from independent concurrently executing finite state machines. The communication and synchronization model is asynchronous, based on message passing over communication channels, which are unbounded buffers. Each communication channel is assumed to be of unbounded capacity, full-duplex, error-free, and based on a FIFO service strategy.

The semantics of a CFSM is defined in terms of a so-called *protocol*. A CFSM protocol is a set of all admissible state sequences. A *global system state* (S, C) of a CFSM protocol is given by the set S of local states of all processes and the status C of all channels, which corresponds to a sequence of messages that have been sent, but not yet received.

CFSM semantics models linear sequences of global system states. The semantics of concurrency is interleaving of transitions of local state machines. Thus, this model is asynchronous and suitable for specification of interactive systems. Considering the expressiveness results, in [BZ83] it is proved that CFSMs are Turing-complete. In consequence of the unbounded capacity of channels, the global state space has unbounded size. Hence, some problems such as termination or reachability of some system state are undecidable. If all the channels of a protocol are defined as bounded then these problems become decidable. Moreover, if all the channels are defined as zero-bounded then the communication model becomes synchronous handshake.

Extended finite state machines

Extended finite state machines EFSMs are Mealy machines extended with data abstraction. A state of an EFSM is given as a function which assign values to data variables. Depending on the kind of transition relation, EFSMs can be both deterministic or nondeterministic. A transition of EFSM has its input side extended with a boolean condition over variables and the output side with operations on variables. The data abstraction is based on the idea of folding of states which are equivalent with respect to boolean conditions into one so-called *symbolic state*.

The formal definition of the EFSM semantics can be found in [CK96]. Since the data space can be potentially unbounded, the state space of EFSM can be infinite. In consequence, considering the expressivity of EFSMs, they are Turing-complete.

Merging the concept of EFSMs and CFSMs, *communicating extended finite state machines* (CEFSMs) were defined and made the foundation for many practical specification and modelling languages – SDL, Estelle and Room/UML-RT. Especially *Specification and Description Language* (SDL), which is standardized by ITU-T in recommendation Z.100, has been widely used in telecommunication industry. The asynchronous communication model of SDL processes is based on potentially unbounded queues. Comparison of various formal semantics for SDL is presented in [Leu94].

Hierarchical finite state machines

Hierarchical finite state machines (HFSM) extend finite state machines with state abstraction and refinement. A HFSM consists of finite state machines with state hierarchy and their parallel composition. The state transitions can be multi-level. The state hierarchy is defined using a single *composition function*, which refines some states with sets of finite state machines. The formal definition of HFSM can be found in [LMS97]. As a graphical formalism, HFSMs combine the concept of *geometric inclusion* (Venn diagrams) with the concept of *hyper graphs* [Har88].

In general, an HFSM merged with the concept of concurrency taken from CFMS is the most succinct state-based formal approach for specification. A lot of visual languages for both interactive and reactive systems are based on these principles. Unfortunately, the complexity of HFSM, i.e. the feature of multi-level transitions, makes achieving of the compositional semantics a very difficult problem. Considering computational expressivity, it depends on the communication model, as in the case of CFMS.

An example of a very complex visual language for reactive systems based on HFSM is the *Statecharts* formalism developed by Harel in [Har87]. Statecharts can be taken as HFSMs with the communication model of instantaneous broadcast. Statecharts in their full strength presents concepts for hierarchy, concurrency (*orthogonal states*), preemption, and history (*history states*). However, while the graphical syntax of the language is so rich, the definition of formal semantics has proved extremely challenging. An example of a simple statechart is in figure 2.2.

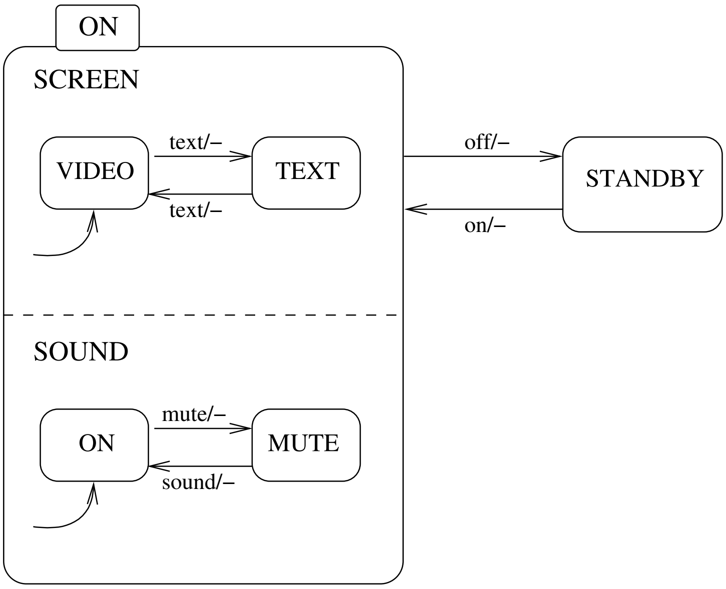

Figure 2.2: Example of a Statechart diagram

The semantic subtlety of statecharts arises from the feature of defining transitions whose enabledness disables other transitions. A statechart may react to an event by performing an enabled transition, thereby performing a so-called *micro step*, which may generate new events that may in turn trigger new transitions while disabling others. When this chain reaction comes to a halt, one execution step, also referred to as a *macro step*, is completed. Together with **compositionality**, there are two other desirable properties of this semantics:

synchrony hypothesis - which guarantees that the reaction of a statechart to a set of input events is maximal and that it terminates before the next set of events enter the system.

causality - which demands that for any event generated at a particular moment, there must be a causal chain of events leading to the action that generated this event. No causal loops can occur and no event may be generated spontaneously.

global consistency - which means that if a constituent micro step of a macro step requires the absence of an event, say a, as one of the conditions for it to trigger, then not only should it not generate a itself, no micro step in the same macro step should generate a.

Requiring of all these properties to be satisfied together raised to some semantic problems. By the result of Huzing and Gerth [HG91], all the desirable properties cannot hold in the same time. One of the first attempts at formalizing the semantics of statecharts through the precise definition of a step was introduced by Pnueli and Shalev in [PS91] with the result of denotational as well as operational semantics satisfying causality, synchrony and global consistency. A formal comparison of some step semantics is presented in [MSPT03]. The first compositional semantics was introduced in [US94]. However, this semantics is not causal. In [LvdBC99] another compositional semantics based on LTS is defined. This result adapts the principle of timed process algebras [HR91], especially the fact that the maximal process assumption is tightly related to the synchrony hypothesis. This operational semantics satisfies all the desirable properties, compositionality holds on the micro step level, while synchrony, consistency and causality are achieved on the macro step level.

To sum up, the various formal semantics of statecharts has been defined, however, only the subdialect of a whole language as proposed by Harel is captured. Especially, the multi-level transitions are not taken into account. Achieving of compositionality for the full statecharts can be done explicitly as in the *Communicating hierarchical state machine* language [RAY99], or implicitly via a synchronization model as in Argos [Mar91], the deterministic and perfectly synchronous variant of statecharts.

Another approach for operational semantics for statecharts was presented by Harel and implemented in a commercial tool STATEMATE [HN96]. The key ideas of this semantics are the following:

1. Reactions to external and internal events, and changes that occur in a step, can be sensed only after completion of the step (inspired with VHDL).

2. Events live for the duration of one step only, the one following that in which they occur, and are not remembered in subsequent states.

3. During the execution of a step, more than one micro step may come from the same parallel component.

4. A maximal subset of non-conflicting transitions is always executed.

5. The execution of a step takes zero time.

Comparing this semantics with the traditional one, the causality property is achieved directly and the global consistency has no sense. Moreover, there are two models of simulation in STATEMATE - the *synchronous* and *asynchronous time* model. The latter satisfies the synchrony hypothesis, while the former does not. This semantics is useful for simulation of practical models, because it supports the very rich version of statecharts equipped with history states, multi-level transitions, etc. On the other hand, neither the syntax, nor the semantics is defined in a compositional manner.

In conclusion, we have to say that there exist a lot of semantical variants of statecharts itself, and additionally, there exist a lot of languages based on the concept of statecharts. Concerning synchronous languages it is especially SyncCharts based on Esterel and Argos, which was already mentioned above. There are also some asynchronous languages based on statecharts, such as GCSR [DS97] or UML-statecharts, which are formalized in [vdB01].

2.2 Data-flow-based methods

Data-flow based visual languages are used for description of process control structures and message exchanges of communicating processes. However, they abstract from internal process behavior. The semantics of a data-flow-based language is based on a particular *coordination model*. Coordination models are build upon a communication media and coordination laws. Examples of coordination media are channels, tuple spaces, buses, etc. Coordination model is given by a number of coordination laws which can describe both asynchronous or synchronous behaviors for communication of processes via coordination media.

Message flow graphs

Message flow graphs (MFGs) are visual notation for describing partial message-passing interaction between communicating concurrent processes.

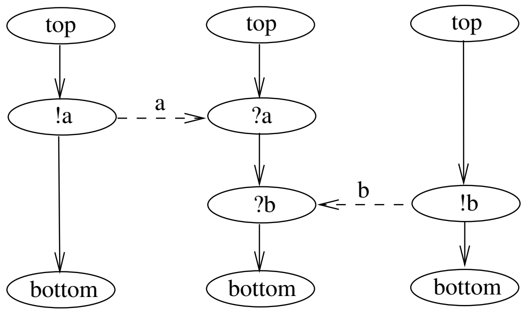

Figure 2.3: Example of an MFG diagram

MFGs may represent different descriptions of communicating processes, e.g. concurrent programming language code, abstract specifications of communication services or protocols, or high level message flow diagrams like the well-known *Message Sequence Charts* (MSCc).

The basic idea of the MFG is that it is represented by a graph structure which is based on the concept of *send* and *receive* events represented as nodes. MFGs have two types of edges, *next-event* and *signal* edges, both represent explicit relations between nodes. Solid arrows represent next-edges and dashed arrows correspond to signal relation. All nodes in MFG, with the exception of the *start* and *finish* nodes, must be connected to just one other node. An example of an MFG is in figure 2.3. The formalization of MFG is given in [Leu94]. To gain more power, MFGs can be equipped with conditions. MFGs, especially MSCs, are used in the description of telecommunication systems, in the analysis of parallel code, and in object-oriented system analysis and design methodologies. An example of the latter are UML-scenarios.

The semantics of MFGs are traces of interleaved atomic communication events. The coordination model can be synchronous or asynchronous. Finite state operational semantics is required for analysis of MFG using formal methods. Finiteness of the global state space can be achieved with the assumption requiring the number of messages being sent at an instant of time to be finite.

Graphical calculus of communicating systems

Another approach aimed to visual specification of coordination aspects of interactive systems with abstraction from internal behavior of components has been proposed in [CDS00]. *Graphical calculus of communicating systems*

(GCCS), as the language is called, adapts the robust process algebraic approach as the formal underlying model. GCCS is based on CCS, which means it is an asynchronous language with the handshake-style coordination model.

The basic principle of GCCS is hierarchical and compositional specification of coordination of components, which have their behavior defined in terms of the process-algebraic operational semantics. Comparing with Statecharts and other state-based languages which support concurrency and hierarchy, the coordination is strictly semantically separated from the behavior of components. Therefore we consider the coordination level of the language as the data-flow-based specification method. This property makes the language an adept for specification of heterogeneous systems.

In GCCS, a component is represented as a *box* with *ports*, which have the meaning of the input/output *interface*. Coordination relations among components are represented as so-called *buses*. The set of boxes interconnected with buses makes a *network*. Networks can be embedded into other networks as components which makes the hierarchical structure. An example of a network is in figure 2.4.

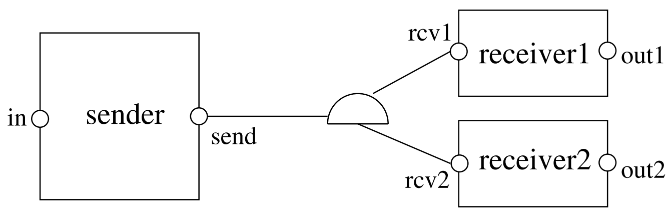

Figure 2.4: Example of a GCCS network

However, GCCS is a framework, and it has to be equipped with more sophisticated constructs to serve as a visual language for complex systems. In [Š01] the precise formalization of GCCS has been done. Also, a simple editor of GCCS with support of transformation of GCCS specifications into CCS has been implemented.

Considering expressiveness of the language, it depends on expressiveness of the language used for the behavioral level. Comparing with MFG, the coordination level itself does not have any executable semantics, more precisely, its semantics is an empty LTS. The semantics of the coordination level take its part in the construction of a resulting LTS by asynchronous composition of LTSs representing the lowest components. In [Š01] it is showed, that this construction corresponds to adding of a bounded number of CCS parallel composition operators.

Synchronous version of GCCS is defined in [Š02]. It is based on SCCS calculus and the construct of buses is extended, so that both synchronous and asynchronous coordination models are combined in one formalism. In [Š03] the strictly synchronous version of the language is defined and the concept of buses is refined to fit the needs of hardware design.

The most recent work on extensions of GCCS is presented in [RC03]. The work has not been published yet and we comment on it with the kind permission of the authors. In the paper, an abstract coordination modelling paradigm called *Architectural Interaction Diagrams* (AIDs) is defined. In principal, an AID is a GCCS network precisely adapted to fulfill the needs of heterogenous high-level design. In the contrary to our previous work, the formal semantics of AIDs is defined directly in terms of labelled transition systems. More specifically, the notion of the input/output labelled transition system is used, so the formalisms like Statecharts or Argos can be easily incorporated into the behavioral level. The concept of buses is extended with ports and state-transition behavior in similar way as we have done it in our algebraic-based semantic approach ([Š02],[Š03]). Buses of AIDs are presented as a framework which allows incorporation of various coordination models, semantically defined as I/O LTSs. The value-passing of AIDs is sketched, but still not precised adequately to fulfill the needs of typed value-passing support.

In conclusion, all the results mentioned above have to be taken as frameworks, not as complex high-level visual languages.

Chapter 3

Aim of the thesis

3.1 Established objectives

The main objectives of this work are the following:

- to explore and evaluate existing semantics of coordination models for reactive and interactive systems
- to propose and develop a visual formal coordination language for specification of complex heterogeneous systems composed from components of both reactive and interactive character
- to justify the heterogeneity of the coordination level by incorporating some highly expressive state-based formalisms into the behavioral level of the developed language
- to implement the user environment (graphical editor) for the developed language with relation to formal verification tools
- to experimentally verify and evaluate the developed tool on practical case-studies

3.2 Already achieved results

In the work [Š01], we have elaborated the idea of a coordination framework [CDS00] of a visual language GCCS for interactive systems based on asynchronous process-algebraic semantics of the CCS calculi. As the behavioral level, we have incorporated finite Moore machines into GCCS.

Considering succinctness, this approach is comparable to communicating finite state machines, with the valuable 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 whole 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 [Š02] 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, which results in a language expressive enough for the specification of systems with both reactive and interactive components. The semantics of the language is defined in terms of SCCS expressions. The concept of buses is refined in such ways that buses serve as active elements which control coordination of components. 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. Moreover, the pure SCCS model is weaker in succinctness than the one used in synchronous languages such as Esterel, because of the unavailability of direct modelling of signal absence. Despite this limitations, the tight connection with the SCCS algebra makes this language useful for visual specification of reactive systems which can be directly formally verified using the concurrency workbench tool.

In [Š03] we have defined the strictly synchronous version of the language and 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 so-called resolution buses.

3.3 Future work

In our future work, we are going to follow the idea [GLL99] of developing the universal coordination framework for different models of computation. However, we primarily focus on mixing of interactive and reactive components. We are going 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 semantical expressiveness of the coordination level of the developed language with other languages.

To achieve the heterogeneity of our language, we would like to incorporate various state-based formalisms to serve for specification of the behavioral level. 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 [KB99] 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 are going 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.

3.4 Progression schedule

- Continuous work on the thesis topic including publications in relevant conferences and workshops.
- Ph.D. final examination - April 2003
- Final version of thesis - December 2004

3.5 Expected outputs

- At least three reviewed publications related to the topic of the thesis.
- A software tool – the implementation of the editor of the developed language available for public.

Bibliography

[APF00] Charles Andre and Marie-Agnes Peraldi-Frati. Behavioral Specification of a Circuit Using SyncCharts: A Case Study. In *Proc. of EUROMICRO'00*. IEEE Computer Society, 2000.

[BBKv02] J. Barnat, T. Brázdil, P. Krčál, and D. Šafránek. Model checking in IPv6 Hardware Router Design. Technical Report 07, CESNET, July 2002.

[Ber98] Gerard Berry. The Foundations of Esterel. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1998.

[BZ83] D. Brand and P. Zafiropoulo. On communicating finite state machines. *ACM*, 30, 1983.

[CDS00] 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.

[CK96] Kwang-Ting Cheng and A.S. Krishnakumar. Automatic generation of functionial vectors using the extended finite state machine model. In *Transactions on Design Automation of Electronic Systems*. ACM, 1996.

[CKS+02] J. Crhová, P. Krčál, J. Strejček, D. Šafránek, and P. Šimeček. Yahoda: the database of verification tools. In *Proc. of TOOLSDAY affiliated to CONCUR 2002*. FI MU report series, 2002.

[CS01] R. Cleaveland and O. Sokolsky. Equivalence and preorder checking for finite-state systems. In *Handbook of Process Algebra*. North-Holland, 2001.

[DS97] I. Lee D.Clarke, H. Ben-Abdallah and O. Sokolsky. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, 1997.

[EMCP99] Orna Grumberg Edmund M. Clarke and Doron A. Peled. *Model Checking*. Cambridge : MIT Press, 1999.

[GLL99] 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.

[Har87] David Harel. Statecharts: A Visual Formalism for Complex Systems. Technical report, The Weizmann Institute, 1987.

[Har88] David Harel. On visual formalisms. *CACM*, 31, 1988.

[HG91] C. Huizing and R. Gerth. On the semantics of reactive systems. Technical report, Eindhoven University of Technology, 1991.

[HN96] David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HR91] M. Hennessy and T. Regan. A process algebra for timed systems. Technical report, University of Sussex, 1991.

[HU79] J. Hopkroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[KB99] M. Koutny and E. Best. Operational and denotational semantics for the box algebra. *Theoretical Computer Science*, 211, 1999.

[Leu94] Stefan Leue. *Methods and Semantics for Telecommunications Systems Engineering*. PhD thesis, University of Berne, 1994.

[LMS97] Yassine Lakhnech, Erich Mikk, and Michael Siegel. Hierarchical automata as model for statecharts. In *Proc. of ASIAN'97*, volume 1345 of *LNCS*. Springer-Verlag, 1997.

[LvdBC99] Gerald Lüttgen, Michael von der Beeck, and Rance Cleaveland. Statecharts via Process Algebra. Technical report, Langley Research Center, 1999.

[Mar91] F. Maraninchi. The Argos language: Graphical Representation of Automata and Description of Reactive Systems. In *IEEE Workshop on Visual Languages*, Kobe, Japan, 1991.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[MSPT03] A. Maggiolo-Schettini, A. Peron, and S. Tini. A comparison of statecharts step semantics. *Theoretical Computer Science*, 290, 2003.

[PS91] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. *Theoretical Aspects of Computer Software*, 526, 1991.

[RAY99] S. Kannan R. Alur and M. Yannakakis. Communicating hierarchical state machines. In *Proceedings of ICALP'99*, volume 1644 of *LNCS*. Springer-Verlag, 1999.

[RC03] Arnab Ray and Rance Cleaveland. Architectural Interaction Diagrams: AIDs for System Modeling. Accepted to ICSE 2003., 2003.

[US94] A. C. Uselton and S. A. Smolka. A Compositional Semantics for Statecharts using Labeled Transition Systems. In *International Conference on Concurrency Theory*, 1994.

[vdB01] Michael von der Beeck. Formalization of UML-Statecharts. In *Proceedings of UML 2001*, LNCS. Springer-Verlag, 2001.

[Š01] David Šafránek. Grafická specifikace souběžných systémů. Master's thesis, Faculty of Informatics, Masaryk University Brno, 2001.

[Š02] David Šafránek. SGCCS: A Graphical Language for Real-Time Coordination. In *Proceedings of FOCLASA'02*, volume 68 of *ENTCS*. Elsevier Science, 2002.

[Š03] David Šafránek. A Graphical Synchronous Language GSCCS. Submitted.
http://www.fi.muni.cz/∼xsafran1/publications/gsccs.ps.gz, 2003.

[Win93] Glynn Winskell. *The formal semantics of programming languages: an introduction*. MIT Press, 1993.

Appendix A

Current state of studies

Completed subjects and courses

During the last two and half years I have studied the field of semantics of reactive and interactive systems as well as the state-of-the-art research results in visual languages for these categories of systems. In addition to the main topic I took several post-graduate subjects in the theory of computer science: *Petri nets*, *λ-calculus*, *Timed automata*, *Semantics of programming languages* and *Modal μ-calculus*.

I also participated on several conferences and summer schools related to the topic:

- BISS 2002 – Bertinoro International Summer School for Graduate Studies in Computer Science, Bertinoro, Italy
- CONCUR 2002 – 13th International Conference on Concurrency Theory, Brno, Czech Republic
- EXPRESS 2002 – 9th International Workshop on Expressiveness in Concurrency, Brno, Czech Republic
- FOCLASA 2002 – 1st International Workshop on Foundations of Coordination Languages and Software Architectures, Brno, Czech Republic
- SOFSEM 2002 – Students Forum, Milovy, Czech Republic

Research

We have successfully formalized and implemented GCCS language [Š01]. Further, we have extended GCCS to its synchronous version based on the SCCS process algebra. The formalization of this extension is done in [Š02]. Our extension has been experimentally used in the design of simple hardware circuits. This experiments have shown that the concept of the language is appropriate for specification of hardware systems, however, we have found a certain lack of succinctness at the behavioral level. Also, it has shown us that the coordination level has to be customized in the sense of extending the model of buses to fit the needs of specification of complex systems. To prove the flexibility of our language, we have customized the synchronous version to fit the needs of hardware and process-control design in [Š03].

In conclusion, our results have to be taken more as frameworks, but not yet as complex high-level visual languages.

Projects

All the research of the author has been done in the projects of the Czech grant agency on the topic concerning algorithms and tools for practical verification of concurrent systems. These are the projects GA201/00/1023 and currently GA201/03/0509.

Moreover, the author participates in the research project of design of *combo6* – IPv6 hardware accelerated card [BBKv02]. This project, which is part of the 6NET (IST-2001-32603) international project, is very interesting due to the fact that it practically motivates our work. In this project hardware designers deal with specification tools for hardware design. Because of the complexity of hardware components, a visual specification tool would make the work of designers more comfortable. In addition, such a tool would bring formal verification methods closer to early design phases, which would be an important contribution to the domain.

The author has also participated on designing the database of verification tools with web interface called YAHODA [CKS$^+$02].

Publications

- D. Šafránek: SGCCS: A Graphical Language for Real-Time Systems, in Mária Bieliková, editor, *Proceedings of Student Research Forum affiliated to SOFSEM 2002 conference*, Milovy, 2002

- D. Šafránek: SGCCS: A Graphical Language for Real-Time Coordination, in A.Brogi, J-M.Jacquet, editors, *Proceedings of 1st International Workshop on Foundations of Coordination Languages and Software Architectures*, ENTCS, volume 68.3, 2002

- J. Barnat, T. Brázdil, P. Krčál, D.Šafránek and V. Řehák: Model checking in the Design of a IPv6 Hardware Router, Technical report 08/2002, CESNET, 2002

- J. Crhová, P. Krčál, J. Strejček, D. Šafránek, and P. Šimeček. YAHODA: verification tools database. In Ivana Černá, editor, *Tools Day, Proceedings*, FIMU-RS-2002-05, pages 99–103. Faculty of Informatics, Masaryk University Brno, 2002. URL: `http://anna.fi.muni.cz/yahoda/`

Presentations

- FOCLASA 2002 – 1st International Workshop on Foundations of Coordination Languages and Software Architectures, Brno, Czech Republic

- SOFSEM 2002 – Students Forum, Milovy, Czech Republic

- Local seminars at the Faculty of Informatics

Others

- Leading the verification group in the project of IPv6 hardware router mentioned above

- Teaching practicals in Introduction to functional programming

- Member of the organization committee of the CONCUR 2002 conference held in Brno, Czech Republic

Appendix B

Summary / Souhrn

Visual methods for specification of software with various design and analysis methods of software engineering, nowadays unified into the Universal Modelling Language UML, appear also useful for specification of reactive and interactive systems. The main advantage of such an universal modelling language is its heterogeneity. Unfortunately, there are no formal languages with appropriate universal character. The objective of the intended thesis is to propose, develop and implement an universal formal visual language for concurrent systems. The most significant properties we are taking into account are the compositionality and hierarchy of the coordination level, which is an important factor in component-based design. Moreover, the design and implementation of the user interface for the developed language is also the aim of the intended thesis.

Grafické metody specifikace softwarových systémů společně s metodami analýzy a návrhu systémů, jež jsou v současné době součástí univerzálního modelovacího jazyka UML, se jeví vhodné rovněž pro specifikaci souběžných systémů. Nejsignifikantnější výhodou univerzálních formalizmů jako UML je jejich vhodnost pro specifikaci systémů složených z komponent heterogenního charakteru. Bohužel dosud nebyly jazyky tohoto typu precizně formalizovány. Účelem zamýšlené disertační práce je navržení a implementace univerzálního jazyka pro grafickou specifikaci souběžných systémů. Nejpodstatnější vlastností, již bereme v úvahu, je kompozičnost a hierarchie koordinační vrstvy navrhovaného jazyka. Tyto vlastnosti jsou klíčové v komponentově orientovaném návrhu systémů. Návrh a implementace uživatelského rozhraní pro navrhovaný jazyk je rovněž součástí navrhované práce.