Statechart Development Beyond WYSIWYG

Steffen Prochnow and Reinhard von Hanxleden

Real-Time and Embedded Systems Group, Department of Computer Science
Christian-Albrechts-Universität Kiel, Olshausenstr. 40, D-24118 Kiel, Germany
`{spr,rvh}@informatik-uni-kiel.de`
WWW home page: `http://www.informatik.uni-kiel.de/rtsys/`

Abstract. Modeling systems based on semi-formal graphical formalisms, such as Statecharts, have become standard practice in the design of reactive embedded devices. Statecharts are often more intuitively understandable than equivalent textual descriptions, and their animated simulation can help to visualize complex behaviors. However, in terms of editing speed, project management, and meta-modeling, textual descriptions have advantages.
As alternative to the standard WYSIWYG editing paradigm, we present an approach that is also graphical but oriented on the underlying structure of the system under development, and another approach based on a textual, dialect-independent Statechart description language. These approaches have been implemented in a prototypical modeling tool, which encompasses automatic Statechart layout. An empirical study on the usability and practicability of our Statechart editing techniques, including a Statechart layout comparison, indicates significant performance improvements in terms of editing speed and model comprehension compared to traditional modeling approaches.

1 Introduction

Statecharts [1] constitute a widely accepted formalism for the specification of concurrent reactive systems. They extend classical finite-state machines and state transition diagrams by incorporating hierarchy, orthogonality, compound events, and a broadcast mechanism for communication between concurrent components. Statecharts provide an effective graphical notation, not only for the specification and design of reactive systems, but also for the simulation of the modeled system behavior. Statecharts have also been incorporated into the Unified Modeling Language (UML) and are supported by several commercial tools, e. g., Rational Rose, Matlab/Simulink/Stateflow, or Esterel Studio. Since the inception of Statecharts some twenty years ago, significant progress has been achieved concerning their semantics, formal analysis and efficient implementation. Concerning the practical handling of Statecharts, however, it appears that comparatively little progress has been made since the very first Statechart modeling tool set [2]. Specifically, the *construction*, *modification*, and *revision management* of Statecharts tend to become increasingly burdensome for larger models, and we feel that in this respect Statecharts are still at a disadvantage relative to other development activities, such as classical programming. This observation, corroborated in numerous discussions with practitioners and modeling experiences ranging from small, academic models to industrial projects, has motivated the work presented in this paper.

A commonly touted advantage of graphical formalisms such as Statecharts is their intuitive usage and the good level of overview they provide—according to the phrase "one picture is worth ten thousand words." However, when moving from toy examples to realistic systems, one is quickly confronted with large and unmanageable graphics originating from a high number of components or from intricate interactions and interdependencies.

As an alternative to the graphical modeling one can also develop reactive systems using textual notations. There exist a couple of languages that either describe Statecharts directly (e. g., SCXML [3], SVM [4]) or indirectly (e. g., Esterel [5, 6]). Consequently the developer of reactive systems may choose between the textual and the graphical approach to specify systems. In principle, they offer the same expressiveness and the same level of abstraction. However, there are notable differences in terms of practical use, and both approaches have their benefits. Graphical models benefit from intuitiveness and are good for higher level context. Textual languages can represent precise details very well and they permit powerful macro capabilities (e. g., using generic scripting or preprocessing languages such as perl or m4) and allow a detailed revision management (e. g., applying the UNIX diff utility to compare different versions).

In summary, textual as well as graphical languages have their specific domains and advantages. The traditional model-based design flow starts with entering a graphical model of the *System Under Development* (SUD), from which textual programs are synthesized; however, as we argue here, it would combine the advantages of both techniques to allow the designer to work with textual and graphical representations of the SUD simultaneously.

Statecharts are commonly created using some *what you see is what you get* (WYSIWYG) editor, where the modeler is responsible for the graphical layout, and subsequently a Statechart appears the way a designer has modeled it. We believe that the WYSIWYG construction paradigm, which leaves the task of graphical layout to the human designer, has so far been a limiting factor in the practical usability of Statecharts, or graphical modeling in general. The premise of this paper is that this paradigm may have been justified at some point, but advances in layout algorithms and processing power today make it feasible to free the designer from this burden.

The main contributions of this paper are:

– an analysis of the graphical editing process using WYSIWYG editors and the identification of generic Statechart editing patterns;
– the presentation of two alternative Statechart construction paradigms—a macro-based and a text-based technique—that let the modeler focus on the modification of the Statechart *structure*, rather than their layout;

– a textual Statechart language, called *KIel statechart extension of doT* (KIT), which is concise and Statechart-dialect independent and supports the text-based construction of Statecharts; and
– an empirical study that evaluates the proposed construction techniques and shows their practicability and efficiency.

The rest of the paper is organized as follows. The remainder of this section discusses related work, and introduces the prototypical *Kiel Integrated Environment for Layout* (KIEL) tool, which serves as an evaluation platform for our proposals. Sect. 2 presents the analysis of WYSIWYG editing patterns. The macro-based and text-based Statechart construction approaches and the KIT language are discussed in Sect. 3. Sect. 4 summarizes the findings of our empirical study, Sect. 5 concludes and discusses possible future extensions.

1.1 Related Work

As indicated above, there is to our knowledge little published work that is directly related to the pragmatics of Statechart construction. However, the work presented here cuts across several related areas that have been studied already, namely textual Statechart description languages, the layout of Statecharts, graphical editors, and cognitive studies on the effectiveness of graphical and textual languages. Each of these areas is briefly discussed in the following.

The SCXML [3] Statechart description language has a comprehensible structure, but the required *tags* and their hierarchical dependencies call for specific XML editors. Alternative Statechart descriptions such as SVM [4] and the UMC [7] Statecharts use explicit declarations of Statechart objects, which reduces the readability especially for large Statecharts. This provides the advantages of textual entry, but does not offer the Statechart dialect-independent, concise constructs as available in KIT. These Statechart description languages generally serve as an intermediate format synthesized from manually edited Statecharts; to our knowledge, none of these languages has been used so far for Statechart synthesis, as we propose to do here. An exception is the RSML approach [8], which synthesizes a graphical view of the topology using a very simple, but surprisingly effective layouting scheme, which inspired KIEL's alternating linear layout. However, RSML still keeps much information that is normally part of the graphical model instead in textual AND/OR tables.

Castelló et al. [9] have developed a framework for the automatic generation of layouts of Statecharts based on floor planning. Harel and Yashchin [10] have investigated the optimal layout of *blobs*, which are edge-less hierarchical structures that correspond to Statecharts without transitions. KIEL offers several layout mechanisms, some employ the GraphViz [11] layout framework, others are developed from scratch.

A well-established technique to obtain consistency between model artifacts produced at different stages of the model life-cycle are transformational modeling approaches. DiaGen [12] and AToM^3 [13] employ graph grammars to generate graphical editors for visual languages. GenGEd [14] uses graph grammars to modify visual languages using graph productions. The visual language is produced by a priori specified production sequences; we here instead propose interactive manipulations of the model. The graph grammar based tools use graphical constraints for placing graphical elements; we perform an automatic layout from scratch.

Several experimental studies address the comprehensibility of textual and visual programs; e. g., Green and Petre [15] performed an experimental study to evaluate the usability of textual and graphical notations using LabView. They determined that visual programs can be harder to read than textual ones. Purchase et al. [16] have evaluated the aesthetics and comprehension of UML class diagrams. We are not aware of any experimental studies on the effectiveness of *editing* visual languages.

1.2 The KIEL Modeling Environment

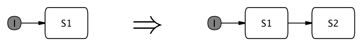
The *Kiel Integrated Environment for Layout* (KIEL) is a prototypical modeling environment that has been developed for the exploration of complex reactive system design [17]. As the name suggests, a central capability of KIEL is the automatic layout of graphical models, which makes KIEL a suitable testbed for the construction paradigms presented here. The following paragraph briefly summarizes KIEL's capabilities.

The tool's main goal is to enhance the intuitive comprehension of the behavior of the SUD. While traditional Statechart development tools merely offer a static view of the SUD during simulation, apart from highlighting active states, KIEL provides a simulation based on the *dynamic focus-and-context* visualization paradigm [17]. It employs a generic concept of Statecharts which can be adapted to specific notations and semantics, and it can import Statecharts that were created using other modeling tools. The currently supported dialects are those of Esterel Studio, Stateflow, and the UML via the XMI format, as, e. g., generated by ArgoUML [18]. Alternatively, KIEL can synthesize graphical SSMs from (textual) Esterel v5 programs [6]. KIEL also provides an automated checking framework, which checks compliance to *robustness rules* [19].

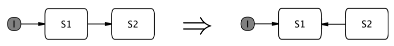
2 The WYSIWYG Statechart Editing Process

To analyze and educe improvements in developing Statecharts, we inspected the common WYSIWYG editing process. We identified nine main *editing schemata*, which can be grouped into three categories: Statechart *creation*, *modification* of Statechart elements, and *deletion* of elements. Fig. 1 illustrates some of these editing schemata. For example, to apply the schema "add hierarchical successor state" (Fig. 1d), the modeler has to perform the following steps: (1) select the state to supplement, (2) add a new hierarchical state, (3) insert an inner initial connector, (4) insert an inner state, and (5) insert connecting transitions.

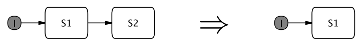
When using conventional Statechart editors, none of the editing schemata can be realized as a single action. Generally, each editing schema using WYSIWYG editors passes the following action sequence:



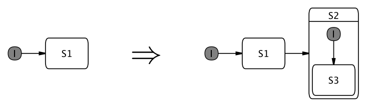
(a) Insertion of a simple successor state.



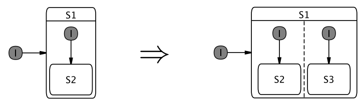
(b) Modification of transition direction.



(c) Deletion of a Statechart element.



(d) Insertion of hierarchical successor state.



(e) Insertion of a parallel region.

Fig. 1: Exemplary generic editing schemata derived from a typical editing process using WYSIWYG editors.

1. If needed, create free space (e. g., expand hierarchical states for new sub-elements, move existing elements for placing new elements).
2. Focus on a Statechart element for modification resp. supplementation (move pointer, select per mouse click).
3. Apply an editing schema.
4. If needed, rearrange the modified chart to improve readability.

It is a common experience that the modeler spends much time with the layout-related activities of steps 1 and 4. For Statecharts developed from scratch, this effort may be small. In contrast, if an existing chart has to be modified, the work for arranging the elements increases roughly with the number of State-chart elements and Statechart complexity. Quoting a practitioner: "*I quite often spend an hour or two just moving boxes and wires around, with no change in functionality, to make it that much more comprehensible when I come back to it*" [20]. Furthermore, each editing schema requires the modeler to perform a sequence of low-level editing steps. The alternative proposals presented in the next section aim to improve both of these points.

3 Proposals for Enhancements in Statechart Editing

The basic idea of our approach is to automate the editing process as far as possible. Specifically, we propose to reduce the effort of re-arranging Statechart elements by applying automatic Statechart layout mechanisms. This produces Statecharts laid out according to a *Statechart Normal Form* (SNF) [17], which is compact and makes systematic use of secondary notations to aid readability. Due to the application of an automatic layout mechanism, the editing action sequence of Sect. 2 is reduced to:

1. Focus on a Statechart element for modification resp. supplementation;
2. Apply an editing schema.

Both editing actions remain under control of the modeler and will be treated by the following editing proposals.

3.1 Macro-Based Modeling

Using WYSIWYG editors, a simple editing action (e. g., placement of a state) scarcely needs time; but applying a complete editing schema (cf. Sect. 2) requires multiple mouse and keyboard actions. Our proposal to optimize this is to directly manipulate the Statechart *structure*, uncoupled from its graphical representation.

The schemata described in Sect. 2 can be interpreted as *Statechart productions*. Before applying a production (a schema), the modeler selects the location for the modification (the focus), which corresponds to the left-hand side of the production. If the production pattern matches, the application of the schema replaces the focus with the right-hand side of the production. The set of productions constitutes a Statechart *grammar*, which has the nice property that every application of a production results in a syntactically correct Statechart. Hence, a design does not go through meaningless intermediate editing stages, which frees the modeler from time-consuming syntax-checking. (An exception to this are productions that delete model elements, which may result in isolated states; KIEL does provide syntax checks that detect these, however.)

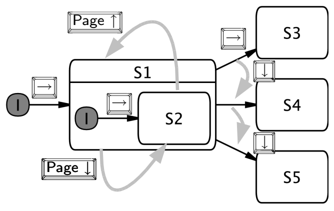
Concerning step 1, the setting of the focus, we propose to not only provide the traditional mouse-oriented mechanism, but also to allow a structure-oriented navigation, similar to text editors. E. g., in the KIEL macro editor (see Sect. 3.3), (1) the right/left key navigates through state sequences, (2) the up/down key navigates among sibling elements (e. g., multiple outgoing transitions from a state object), and (3) the page up/down keys navigate up resp. down in state hierarchies. Fig. 2a illustrates some navigation examples.

Concerning step 2, the selection of an editing schema, the designer may select a schema from a pull-down menu or by pressing a keyboard shortcut. E. g. in KIEL Ctrl+I generates a new successor state with a connecting transition and adjusts if necessary the priorities associated with transitions (cf. Fig. 2b). Afterwards a rearrangement of the Statechart elements will be performed automatically, according to the SNF.

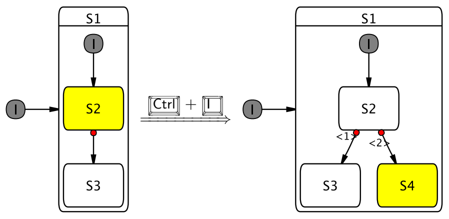
3.2 Text-Based Modeling

Macro-based modeling works directly on the Statechart topology, combining several simple editing actions. As another, alternative structure-based Statechart editing technique we propose to employ a textual Statechart structure description. The *KIel statechart extension of doT* (KIT) combines implicit declarations as used in *dot* [11], the hierarchy construction as used in textual Argos [21], and the orthogonal construction as used in Esterel [22] with the ability to describe different dialects of Statecharts.

Fig. 3 presents a KIT example with the equivalent graphical model of a Safe State Machine (SSM), the Statechart-dialect implemented in Esterel Studio. Fig. 3a lists the KIT code, which is shortly described in the following. The Statechart preamble is listed in Line 1, containing the Statechart name and the model type and version, which determine the Statechart dialect and the accompanying



(a) Navigation with key strokes.

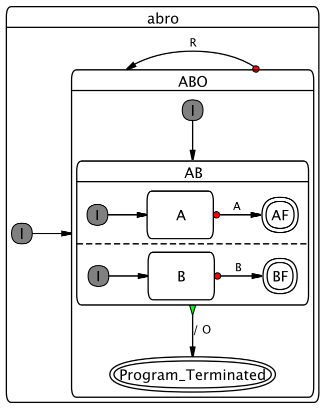


(b) Example of applying the “insert simple successor state” schema. Before applying the schema, state S2 is selected; afterwards the inserted state S4 remains selected for further editing operations.

Fig. 2: Editing actions and navigation using the macro-based modeling approach.

```
 1  statechart abro[model="Esterel Studio";version="5.0"]{
 2   input A;
 3   input B;
 4   input R;
 5   output O;
 6   {
 7    ->ABO;
 8    ABO{
 9     AB{
10       ->A;
11       A->AF[type=sa;label="A"];
12       AF[type=final];
13      ||
14       ->B;
15       B->BF[type=sa;label="B"];
16       BF[type=final];
17     };
18     ->AB;
19     AB->Program_Terminated[type=nt;label="/ O"];
20     Program_Terminated[type=final];
21    };
22    ABO->ABO[type=sa;label="R"];
23   };
24  };
```

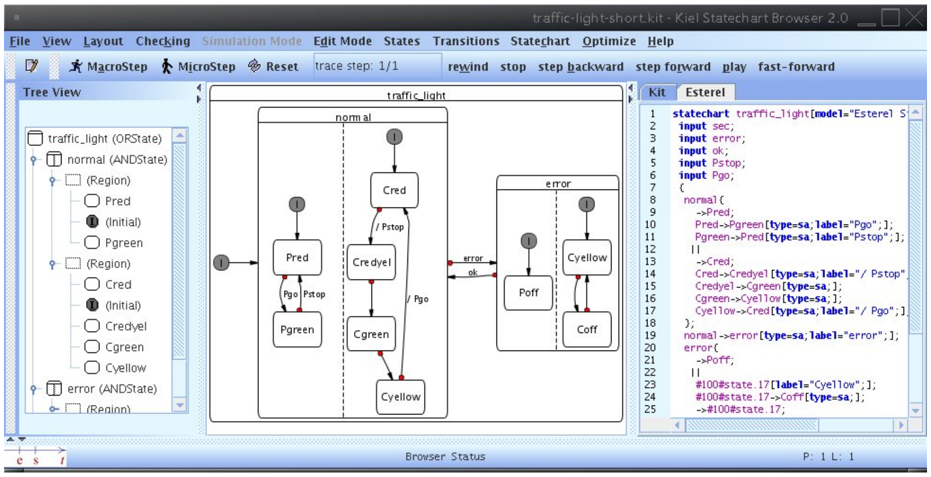
(a) KIT description representation.



(b) SSM representation.

Fig. 3: Textual and graphical representations of the ABRO example [22].

graphical Statechart representation of the targeted modeling tool. Lines 2–5 declare the signal events. Afterwards, Lines 7–23 declare Statechart elements and their relations. State objects are implicitly identified by their state names, (cf. Line 8), curly braces define the scope of hierarchical relations (e. g., state AB, cf. Line 9–17), transitions are written as -> (cf. Line 11), and the || operator denotes parallel regions (cf. Line 13). KIT includes a couple of shorthand notations; e. g., a transition without a source node determines an initial connector (cf. Line 7), a transition of type sa abbreviates the SSMs *strong abortion* (cf. Line 11).



```
statechart traffic_light[model="Esterel S
 input sec;
 input error;
 input ok;
 input Pstop;
 input Pgo;
 {
  normal{
    ->Pred;
    Pred->Pgreen[type=sa;label="Pgo";];
    Pgreen->Pred[type=sa;label="Pstop";];
   ||
    ->Cred;
    Cred->Credyel[type=sa;label="/ Pstop"
    Credyel->Cgreen[type=sa;];
    Cgreen->Cyellow[type=sa;];
    Cyellow->Cred[type=sa;label="/ Pgo";]
  };
  normal->error[type=sa;label="error";];
  error{
    ->Poff;
   ||
    #100#state.17[label="Cyellow";];
    #100#state.17->Coff[type=sa;];
    ->#100#state.17;
```

Fig. 4: Screenshot of KIEL displaying the Statechart tree-structure, the graphical model, and the KIT editor.

3.3 Implementation in KIEL

We have implemented the above proposed Statechart editing techniques in KIEL, which resulted in the *KIEL macro editor* and the *KIT editor*. Both editors are accessible simultaneously and are arranged side by side so that they allow alternative views on the same SUD, as can be seen in Figure 4. The user may thus chose to manipulate either the textual or the graphical view, and the tool keeps both views automatically and continuously consistent.

The KIEL macro editor is implemented as an extension of the graphic displaying window; there the modeler marks and modifies graphical elements directly. The KIT code is kept in sync with the graphical model. In the opposite direction, if the modeler is using the KIT editor, the graphical model is synthesized from KIT code. This employs a parser/synthesizer generated by SableCC [23]. A similarly generated tool performs the application of the production rules using the KIEL macro editor. The productions are specified using an underlying grammar; the set of production can be easily extended with further production rules. Figure 5 depicts the tool chain of the KIT editor and the KIEL macro editor and their integration within KIEL.

4 Experimental Evaluation

We have used successive generations of KIEL in the classroom since 2005, including the macro-based and text-based editors presented here. The feedback on these editing approaches has been generally quite positive, also in comparison to the classical editing paradigms employed by the other, commercial modeling tools also used in classes. However, to gain a better, objective insight into the

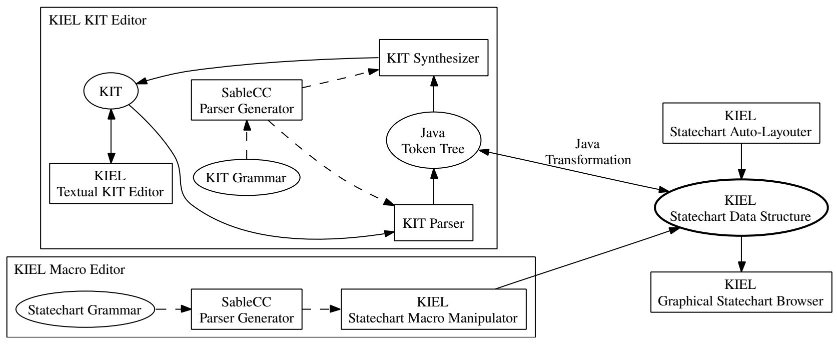


Fig. 5: Integration of the KIT editor and the KIEL macro editor into KIEL. The solid lines characterize the information flow during runtime, the dashed lines represent dependencies during compile-time of KIEL.

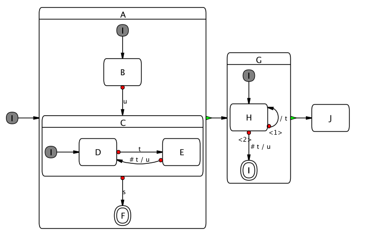
effectiveness of our modeling approaches, we have performed an experiment to investigate differences in editing performance between the conventional WYSIWYG approach, the KIT editor, and the KIEL macro editor. As mentioned in Sect. 3.3, KIEL provides a mechanism that automatically produces our preferred Statecharts arrangements; in the following we call this the *alternating dot layout* (ADL) (see Fig. 6a). Hence, a further goal of the experiment was to compare the readability of the ADL with other layout strategies.

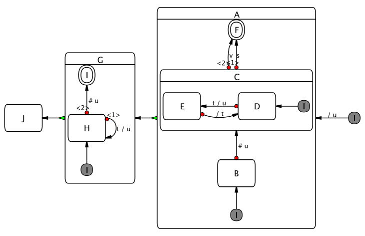
4.1 Experiment design

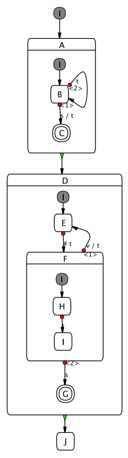
The participants in the experiment (the *subjects*) were graduate-level students attending the lecture "Model-Based Design and Distributed Real-Time Systems" in the Winter Semester 2006/07[1]. Most of them were not familiar with the Statechart formalism in advance. The experiment consisted of two parts. The first part took place early in the semester, after two lecture units introducing the Statechart formalism. The subjects had by then also solved a first homework on understanding Statechart semantics. The second part proceeded after the final lecture unit at the end of the semester. In the meantime the subjects had gained practical experiences in modeling Statecharts; furthermore, they had learned about the importance of modeling paradigms, such as maintainability and co-notation of Statecharts. 24 students participated in the first experiment, 19 took the second one. In the following we refer to the participants of the first experiment as *novices* and to the participants of the second experiment as *advanced*. Furthermore, we define as *experts* the modelers that have significant practical experience beyond course work. Both experiments have similar design and consist of three parts:

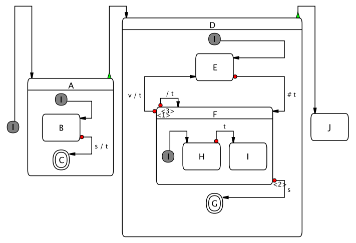
Modeling Technique Evaluation: The subjects had to create Statecharts of varying complexity using different Statechart modeling techniques: a graphical

[1] URL: `http://www.informatik.uni-kiel.de/rtsys/teaching/ws06-07/v-model/`


(a) Alternating dot layout (ADL).


(b) ADL backwards (ADBL).


(c) Linear layer layout (LLL).


(d) Alternating linear layout (ALL).

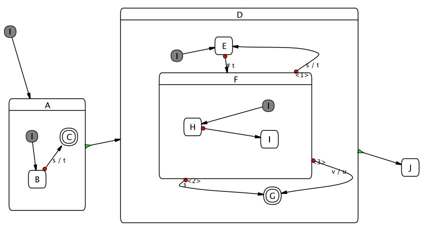

(e) Arbitrary layout (AL).

Fig. 6: Different Statechart layouts for experimental comparison. The layouts in Fig. 6a, 6b, and 6d were automatically generated from the KIEL layout mechanism; the Statechart models in Fig. 6c and 6e were drawn manually.

WYSIWYG Statechart editor (we decided to employ Esterel Studio), the KIEL macro editor, and the KIT editor. Afterwards the created Statechart had to be extended and modified. A one-page reference card per modeling tool instructed the subjects. As editing performance metric the elapsed time was measured.

Subjective Layout Evaluation: The subjects were asked to score readability and comprehensibility of five different Statechart layouts without understanding (cf. Fig. 6).

Objective Layout Evaluation In this experiment the subjects had to analyze different Statechart models, constructing a sequence of active states and upcoming signal events according to the semantics of SSMs. The elapsed time of each Statechart reading was measured for performance evaluation.

Each subject had to process a personal randomized experiment assignment (see Sect. 4.3), containing the tasks described above. The study was realized as a controlled experiment, i. e., the experiment leader checked and rejected the solutions of Parts A and C in case of incorrectness. Each of the subject's experiments was performed in a single session of one to two hours; the sessions were videotaped.

4.2 Hypotheses

The main questions asked in this experiment are the following: "Do the macro-based and text-based editing techniques make the Statechart construction process easier and faster than the conventional WYSIWYG method? Are the resulting Statecharts more readable and comprehensible?" To guide the analysis of the results it is useful to formulate some explicit expectations in form of hypotheses about the differences that might occur. Hence, the experiment should investigate the hypotheses as follows:

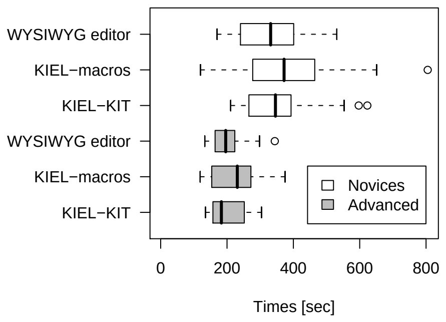
1. *Statechart Creation:* We expect that novices will need less time to create a Statechart using the WYSIWYG editor compared to the usage of the KIEL macro editor or the KIT editor. However, the Statechart creation times of advanced modelers using the KIT editor should be less than when using the WYSIWYG editor.
2. *Statechart Modification:* We expect that modification of an existing Statechart using the KIT editor or the KIEL macro editor is faster than using the WYSIWYG editor.
3. *Aesthetics:* Statecharts are sensed as aesthetic if their elements are arranged conforming to a certain layout style guide. We expect the best scores for Statecharts laid out according to the ADL (see Fig. 6a).
4. *Comprehension:* We suppose that well arranged Statecharts influence the readability. Hence, we expect a faster comprehension of the ADL compared to other Statechart layouts.

4.3 Validity

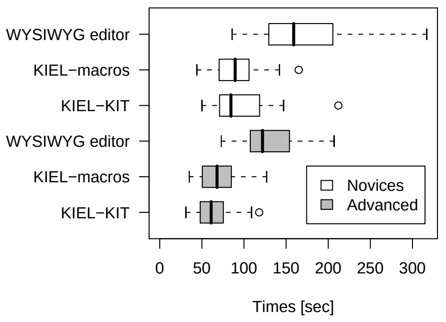
Concerning the *internal validity*, all relevant external variables (subjects' Statechart modeling experience, maturation, aptitude, motivation, environmental condition, etc.) were equalized between appropriate groups by randomized group assignment. Regarding the *external validity*, there are several sources of differences between the experimental and real Statechart modeling situations that limit the generalization of the experiments: In real situations, there are modelers with more experience, often working in teams, and there are Statechart models of different size or structure. However, we do not consider this to invalidate the basic findings of the experiment.

4.4 Results

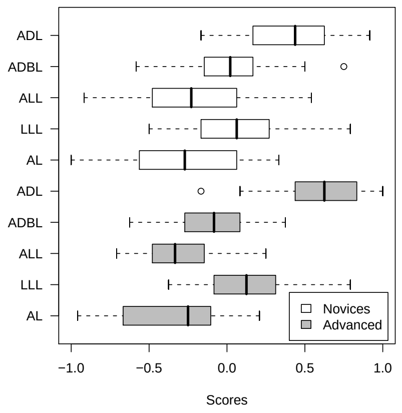
This section presents and interprets the results of the experiments. The analysis is organized according to the hypotheses listed in Sect. 4.2. *Box plots* present the obtained statistical data; the comparison of means will be assisted by the two sample *t-test*. The test compares the difference of sample means from two data series to an hypothesized difference of the series means. It computes the p-value, which indicates statistically significance. We will call a difference significant if $p < 0.05$. The analysis and plots were performed with R v. 2.4.0 [24].



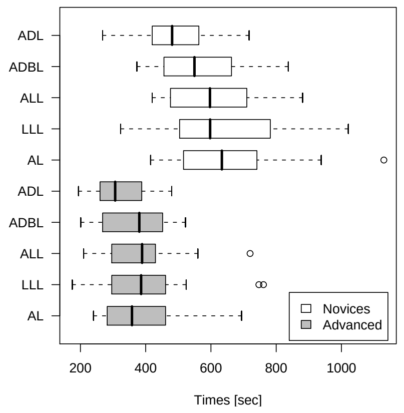
(a) Distribution of times for creating a new Statechart. *Novices:* The Statechart creation times using a WYSIWYG editor are smaller than using the KIEL macro editor (t-test $p = 0.04$) and tend to be smaller using the KIT editor (t-test $p = 0.25$). *Advanced:* The Statechart creation times using a WYSIWYG editor tend to be smaller than using the KIEL macro editor (t-test $p = 0.12$); time differences between WYSIWYG editor and KIT editor are not significant (t-test $p = 0.46$).



(b) Distribution of times for modifying an existing Statechart. *Novices and Advanced:* The needed times for Statechart modification using the KIEL macro editor or using the KIT editor are smaller than the times using the WYSIWYG editor (both: t-test $p = 0.00$).



(c) Distribution of subjective Statechart layout scores. *Novices and Advanced:* The ADL scores better than all other Statechart layouts (all layouts: t-test $p = 0.00$). Score meaning: 1.0: strong preference, -1.0: strong rejection.



(d) Distribution of Statechart comprehension times. *Novices:* Less time is needed for comprehending Statechart according to ADL (ADBL: t-test $p = 0.02$, others: t-test $p = 0.00$). *Advanced:* ADL times tend to be smaller than times of the ADBL (t-test $p = 0.1$); less time is needed for other layouts (ALL: t-test $p = 0.04$, LLL: t-test $p = 0.03$, AL: t-test $p = 0.03$).

Fig. 7: Distribution of times for modeling Statecharts and distribution of Statechart layout assessments. The box plots denote quartiles, small circles indicate outliers.

Evaluation of Modeling Techniques. The plots in Fig. 7a corroborate our Hypothesis 1 for novices. Due to the novelty of the KIEL macro editor and KIT editor, the novices sought advice in the reference card (cf. Sect. 4.1); in contrast the WYSIWYG editor could be used intuitively and without any reference card. Hence, on average the novices needed less time for creating Statecharts using the WYSIWYG editor than using the KIT editor or the KIEL macro editor. For advanced learners, however, the mean times are slightly less using the KIT editor. We suppose that for experts in Statechart creation this difference would increase further.

Fig. 7b illustrates the efficiency using the KIT editor and the KIEL macro editor in Statechart modification; this corroborates Hypotheses 2. With the KIT editor and the KIEL macro editor the modeler only works on the Statechart structure, while KIEL's Statechart auto-layouter arranges the graphical model. In contrast, using the WYSIWYG editor the subjects spent most of the time with making room for new Statechart elements and re-arranging the existing ones to make the developed chart readable. Despite the fewer operations needed using the KIEL macro editor, the subjects needed more time to modify a Statechart. The time was largely due to frequent consultations of the reference cards. Hence, for experts we suppose that the KIEL macro editor would provide the fastest modeling method.

Evaluation of Statechart Layouts. The scores of subjective Statechart layout assessment (cf. Fig. 7c) clearly show the subjects' preference for Statecharts laid out according to the ADL; hence, hypothesis 3 can be retained. Apparently it is not sufficient that layouts underlie an automatic layout; in fact Statechart layouts have to satisfy certain aesthetics to be assessed as good layouts. Accordingly, subjects stated that "transitions must be short and traceable" and "the element structure has to follow the Statechart meaning". E. g., due to unnecessary long transitions the ALL scores lower than the LLL.

Figure 7d demonstrates that a proper layout enhances the readability of Statecharts; Statecharts laid out according to the ADL are faster comprehensible than other Statechart layouts, which corroborates hypothesis 4. This results from the accompanying proper micro layout (e. g., label placement) as well as proper macro layout (e. g., compact and white-space avoiding element arrangement).

5 Conclusion and Future Work

Embedded devices are proliferating, and their complexity is ever increasing. Statecharts are a well established formalism for the description of the reactive behavior of such devices. However, there is evidence that the current use of this formalism is not optimal, in particular as models get more complex.

We have presented a description language called KIT that was developed with the intention to describe topological Statechart structures. The KIEL tool combines the ability of easy textual editing and simultaneous viewing of the resulting graphical Statechart model. As another alternative to the classic, low-level WYSIWYG graphical editing paradigm, the graphical model can be modified using high-level editing schemata. This technique employs Statechart production rules that ensure the syntax-consistency through the whole editing process. The user feedback on this has been generally very positive, and this has been supported by experimental data.

In the future we intend to experiment further with the simultaneous display of textual and graphical representation of the SUD. E. g., for a better traceability an indexing mechanism between elements of the textual and the graphical models could be useful. Beyond, we intend to apply the graphical model synthesis from a textual description, in combination with layout and simultaneous display, to data-flow languages such as SCADE/LUSTRE.

Acknowledgments. Mirko Wischer has performed the implementation of the KIEL macro editor and KIT editor presented here; the rest of the KIEL development team have also contributed. Prof. Dr. Jürgen Golz, Dept. of Psychology, CAU Kiel, has been an invaluable help in the design and analysis of the validation experiment. We also thank Ken Bell and the anonymous reviewers for helpful comments on this paper. Finally, we thank the students participating in the experimental study for their support and interest.

References

1. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8**(3) (June 1987) 231–274
2. Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., Trakhtenbrot, M.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems. IEEE Transactions on Software Engineering **16**(4) (April 1990) 403–414
3. W3C: State Chart XML (SCXML): State Machine Notation for Control Abstraction (February 2007) URL: `http://www.w3.org/TR/scxml/`.
4. Feng, T.H.: An extended semantics for a Statechart Virtual Machine. In Bruzzone, A., Itmi, M., eds.: Summer Computer Simulation Conference (SCSC 2003), Student Workshop, The Society for Computer Modelling and Simulation (July 2003) S147—S166 Montréal, Canada.
5. Berry, G., Gonthier, G.: The Esterel Synchronous Programming Language: Design, Semantics, Implementation. Science of Computer Programming **19**(2) (1992) 87–152 URL: `http://citeseer.nj.nec.com/berry92esterel.html`.
6. Prochnow, S., Traulsen, C., von Hanxleden, R.: Synthesizing Safe State Machines from Esterel. In: Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'06), Ottawa, Canada (June 2006)
7. Mazzanti, F.: UMG User Guide (Version 2.5). Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" (ISTI), Pisa, Italy. (2003)
8. Heimdahl, M.P.E., Leveson, N.G.: Completeness and Consistency in Hierarchical State-Based Requirements. Software Engineering **22**(6) (1996) 363–377

9. Castelló, R., Mili, R., Tollis, I.G.: A Framework for the Static and Interactive Visualization for Statecharts. Journal of Graph Algorithms and Applications **6**(3) (2002) 313–351
10. Harel, D., Yashchin, G.: An Algorithm for Blob Hierarchy Layout. The Visual Computer **18** (2002) 164–185
11. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Software—Practice and Experience **30**(11) (2000) 1203–1234 URL: `http://www.research.att.com/sw/tools/graphviz/GN99.pdf`.
12. Minas, M.: Specifying Statecharts with DiaGen. HCC '01 – 2001 IEEE Symposia on Human-Centric Computing Languages and Environments, Symposium on Visual Languages and Formal Methods, Statechart Modeling Contest (September 2001) URL: `http://www2.informatik.uni-erlangen.de/VLFM01/Statecharts/minas.pdf&e=747`.
13. de Lara, J., Vangheluwe, H., Alfonseca, M.: Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. Software and Systems Modeling (SoSyM) **3**(3) (August 2004) 194–209
14. Bardohl, R.: GenGEd – A visual environment for visual languages. Science of Computer Programming, Special Issue of GraTra '00 (2002)
15. Green, T.R.G., Petre, M.: When Visual Programs are Harder to Read than Textual Programs. In: Human-Computer Interaction: Tasks and Organisation, Proceedings ECCE-6 (6th European Conference Cognitive Ergonomics). (1992) URL: `http://citeseer.nj.nec.com/green92when.html`.
16. Purchase, H.C., McGill, M., Colpoys, L., Carrington, D.: Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study. In: ACM International Conference Proceeding Series archive, Australian symposium on Information visualisation. (2001) 129–137
17. Prochnow, S., von Hanxleden, R.: Comfortable Modeling of Complex Reactive Systems. In: Proceedings of Design, Automation and Test in Europe (DATE'06), Munich (March 2006)
18. ArgoUML: Tigris.org. Open Source Software Engineering Tools URL: `http://argouml.tigris.org/`.
19. Prochnow, S., Schaefer, G., Bell, K., von Hanxleden, R.: Analyzing Robustness of UML State Machines. In: Proceedings of the Workshop on Modeling and Analysis of Real-Time and Embedded Systems (MARTES'06), held in conjunction with the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS/UML 2006, Genua (October 2006)
20. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Communications of the ACM **38**(6) (June 1995) 33–44
21. Maraninchi, F.: The Argos language: Graphical representation of automata and description of reactive systems. In: IEEE Workshop on Visual Languages. (October 1991)
22. Berry, G.: The Foundations of Esterel. Proof, Language and Interaction: Essays in Honour of Robin Milner (2000) Editors: G. Plotkin, C. Stirling and M. Tofte.
23. Gagnon, E.M., Hendren, L.J.: SableCC, an object-oriented compiler framework. In: TOOLS (26), IEEE Computer Society (1998) 140–154
24. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. (2006) URL: `http://www.R-project.org`.