Proceedings of the IASTED International Conference
Applied Modelling and Simulation
September 1-3, 1999, Cairns, Australia

Modeling Adaptive Reactive Systems

João José Neto, Jorge Rady de Almeida Júnior
Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Computação e Sistemas Digitais
e-mails: jjneto@pcs.usp.br , jrady@pcs.usp.br

Keywords: Adaptive Models, Adaptive Statecharts, Systems Modeling

Abstract

Reactive systems have been described by means of several notations and formalisms, through which different orthogonal aspects are captured: their internal state behavior, their synchronization aspects and their communication component. Classical formalisms are intended in a large extent to capture static aspects of systems behavior. However, for systems which change dynamically in response to external stimuli, that approach is not enough to represent all its features, because of the lack of ability of classical models to easily describe dynamic changes in the systems behavior.

In order to fulfill this requirement, an adequate model for representing dynamically changing systems should be based on some kind of formal device having the feature of representing self-modifications. We call *adaptive* such a class of formal devices, and the technology they allow to implement. In the following sections some adaptive devices are described as a qualitative introduction to the modeling of adaptive reactive systems.

As an illustration, a very simple reactive system with adaptive properties is described by means of adaptive statecharts: an interactive system for decompressing 2-D packed sequences based on geometric shapes moving on a plane allowing the user to disturb the exhibition of the sequence by modifying shape locations during its presentation.

In this paper we explore the concepts of adaptive technologies applied to the modeling of reactive systems. In particular, adaptive automata and adaptive statecharts are considered. [1], [8]

Adaptive Automata

Adaptive automata derive from the conceptual difference between context-free and context-sensitive languages: in contest-sensitive languages, the occurrence of some particular construct in some specified environment may imply modifying the way related constructs are to be interpreted throughout of the input, and that may be obtained by modifying the behavior of the acceptor for that language.

A very elegant and strictly syntactic way to modify the interpretation of the input is to modify the syntax the automaton is able to accept, so fitting the needs of the new situation.

A straightforward method for modifying the syntax accepted by an automaton is to change its topology according to the desired syntax modifications.

Any changes to an automaton may be accomplished through a set of elementary editing operations acting upon its set of transitions: adding a new transition to the set, selecting from the set of transitions a target transition that fit some given predicate, and eliminating a selected transition from the set.

Editing operations may be grouped, and the resulting groups are named adaptive actions.

Adaptive automata are built from subjacent pushdown automata simply by attaching adaptive actions to their transitions. That enables them by applying adaptive actions attached to their state transitions.

An editor and a simulator have been implemented for adaptive automata, allowing them to be tested and debugged in a friendly environment.[4]

Adaptive automata have been used as the starting point to a number of other correlate developments, both in their theoretical and practical issues, giving rise to variants as Adaptive Statecharts and Synchronized Adaptive Statecharts [8] (see below).

Adaptive automata have the power of a Turing Machine but, in contrast to this classical formalism, it is sufficiently attractive and effective for being used in an easy way by implementers as a specification language, because it behaves as a formal description meta-language that allow the natural construction of practical and efficient implementations.

By observing the operation of adaptive automata, a single unusual feature may be indicated to be responsible by most of their practical power, namely its input-driven self-modifying capacity.

This adaptive paradigm is not exclusive to adaptive automata, but it may be applied as well to several other originally static formalisms, giving rise to a growing family of powerful dynamically modifiable formal models. Some other models in this class of devices have been reported in the literature [2],[5],[6],[7]

Adaptive Statecharts

By attaching adaptive features to statecharts, the resulting model was called the adaptive statechart.

Adaptive statecharts are based on a variant of standard statecharts, and were designed to be used for describing reactive systems.

Their main features include statecharts hierarchical high-level graphical notation, allowing it to be employed for analysis and design purposes.

Adaptive features added to the classical model enable it to dynamically change its own behavior. The resulting software is a powerful tool for describing reactive systems with learning power.

This work originated also a visual tool for editing and simulating adaptive statecharts, which was conceived and developed as a part of a doctoral research [3].

Currently this tool is being fully reprogrammed in order to improve its facilities and to make better its man-machine interface for use by non-expert people.

Adaptive Statecharts have been employed in projects on reliability and security analysis of transport systems, like railways or metropolitan subway systems

Synchronized Adaptive Statecharts

Synchronized Adaptive Statecharts are hybrid models that were created as extensions of adaptive statecharts through the addition of explicit synchronizing features to the model.

Although adaptive statecharts use the implicit communicating and synchronizing features of classical statecharts, synchronized adaptive statecharts allow explicitly stated synchronization among statecharts through connecting elements based on restricted-shaped Petri Nets.

The resulting model allow factoring all useful synchronizing aspects that seem to be relevant to the analysis by extracting them from the statecharts and migrating them into equivalent elements in the connecting Petri Net elements.

So, all asynchronous phenomena not important to the analysis may remain hidden as internal statechart elements, and only those really needed for the current study may be converted into Petri Net connections for better monitoring purposes.

It is possible to derive the Petri Net connections automatically from the original statecharts by selecting the events that are to be observed through the connecting elements, but no provision for that had been implemented yet.

An extension to the previously mentioned tool for adaptive statecharts has been implemented, allowing the use of the tool to develop and debug synchronized statecharts too. [8]

Motivation

Using adaptive automata or adaptive statecharts is particularly adequate for expressing context-dependent user-oriented applications with state-based internal structure and learning abilities.

In order to implement any adaptive machine, one may start from an initial non-adaptive device of the same kind and attach to it rules for input-driven dynamic self-modification.

For that purpose, it is needed some mechanism allowing to inspect the contents of the system memory, as well as its set of states and its input stream. Furthermore, this device must be able to generate new rules updating the system's description.

Adaptive automata and adaptive statecharts may be employed with that purpose. Although finite-state automata, pushdown automata and standard statecharts are unable to handle context-dependencies and self-modification, they are indeed used as the underlying static devices for adaptive automata and adaptive statecharts, respectively.

Therefore, time, input and output memories and system states are the basic elements adaptive automata and adaptive statecharts are intended to handle.

The complexity of an application may be roughly evaluated taking into account the number of its variables, the number of structural levels in its architecture and the freedom of choice of its configuration parameters.

Some typical applications for adaptive devices are in the field of formal languages, process control, computer art, robotics and artificial intelligence, including machine learning, computer education, optimization, handicapped people training, natural language processing, simulations and games, children education, dynamic traffic control, and many other intelligent or behavior-changing systems.

Using Adaptive Technology for Viewing Packed 2-D Sequences

This example illustrates a 2-D viewer for packed sequences of moving geometric shapes. We assume in this paper that a compressor program is run previously for building the packed form of the movie from the original film, by considering in the packed form the minimum set of frames that properly represent the movement of the set of geometric shapes.

The compressing technique consists of representing the original movie by a subset of its original frames without loosing information. That is achieved by splitting the original film into a sequence of takes. In each take, each shape is allowed to move from an initial position to a final one through a straight path, with constant speed. With that restriction, any take may therefore be packed as a pair of frames, stating the original and final positions of the geometric shapes, and the time to elapse between these two situations. Packing is further improved by coding the shapes by specifying their types and attributes, as in ordinary vector imaging.

In order to watch to a movie packed this way, one should run some viewer capable to perform its decompression. Such an operation is quite trivial, consisting on extracting sequentially the pair of frames that describe each take and reconstructing the intermediate frames by interpolating the shapes along the corresponding straight path from its initial position towards its final position, according to the duration specified for the take.

Note that no interactions have been specified between the movie and the external environment, so all the operations described above may be easily performed without using adaptive devices.

However, in our illustrating interactive example, during the exhibition of the movie, the user is allowed to disturb its presentation by dragging shapes apart from their path. Although the current position of the shape is changed by that action, the final position of the shape in its original path will be maintained by the viewer, and another path from the new position to the original final position is drawn, so the shape starts to move through the new path towards the final position.

Although the current implementation is restricted to displacement actions, other disturbances would be allowed easily in the system, such as modifications in rotation speed, size variations and dynamic color changing for the shapes.

Other improvements may be included in this system in a second moment, regarding the achievement of special effects through specification of more complex run-time behavior defaults, such as non-straight paths, non-uniform speed for the shapes, variable-interval interpolation for the intermediate frames, fading effects for the take, insertion of sound effects, etc.

Our Adaptive 2-D Presentation System consist of a compressor that builds the packed form of the movie from an original non-compressed movie (or an film editor, that generates the compressed movie directly from user specification), and a corresponding viewer, which performs the needed interpolations in order to restore and present the packed movie in visual form. The overall architecture of our system is sketched in the figure below.

$$\begin{tikzcd}
\textbf{Original frame sequence} \arrow[rr] \arrow[d] & & \textbf{Restored frame sequence} \\
\textbf{editor (compressor)} \arrow[r] & \textbf{packed movie} \arrow[r] & \textbf{viewer (expander)} \arrow[u]
\end{tikzcd}$$

Specification of the System

We start the description of our system by choosing some notational framework through which we will represent movies in packed form, as described before.

- a packed movie PM may be specified as a sequence of frame pairs FP_i , composed of its left frame LF_i and its right frame RF_i .

$$PM = FP_1\ FP_2\ FP_3\ \dots\ FP_n$$
$$FP_i = LF_i\ \ RF_i\ ,\ 1 \le i \le n$$

- When PM is viewed, a sequence SF_i of interpolated frames replace the corresponding pair FP_i in the packed form of the movie, and the restored movie RM will be the resulting sequence of frames. Interpolation is performed so that all resulting frames are equally spaced by some adequately chosen refresh time rt along the duration time Δt_i of the take associated to FP_i

$$RM = SF_1\ SF_2\ \dots\ SF_n$$
$$SF_i = F^i{}_1\ F^i{}_2\ \dots\ F^i{}_{m_i}\ ,\ 1 \le i \le n, \quad \text{with}$$
$$LF_i = F^i{}_1 \ \text{ and } \ RF_i = F^i{}_{m_i} \quad \text{and} \quad m_i = 1 + \lceil \Delta t_i / rt \rceil$$

- Each frame $F^i{}_k$ ($1 \le i \le n$, $1 \le k \le m_i$) in RM may be described as a list of components. Each component is represented by the corresponding elementary geometric shape type and its set of attributes (depending on the type of the component, attributes vary in number and nature)

$F^i{}_k$ are of the form

(*identification*, *type*, *attribute*, *attribute*, ...) where

identification – is some code that designates the shape univocally

type – stands for the nature of the geometric shape: straight segment, circle, etc.

attribute – stands for x- and y- coordinates, color, layer, thickness, etc.

- For dynamic operations, translation, zooming, rotations and fading effects are allowed.
- At any moment, while the movie is exhibited, the user may dynamically introduce external disturbances to arbitrarily chosen patterns of the current frame by dragging, imposing rotation, zooming or color changing.
- defaults may be established for all parameters. If nothing else is specified, defaults are used: minimum rotation angle, number of intermediate frames, straight path, steady speed, constant color, minimum and constant angular speed

From a dynamic viewpoint, the operation of our system proceeds as follows:

- establish an adequate number of intermediate frames for the next pair of frames in the packed movie.
- establish the space and color path to be followed by each moving element or group in the scene.
- for each intermediate frame, calculate the adequate values of the attributes for all elements in the scene, throughout their respective paths

- extract sequentially the frames from the expanded movie and draw all its elements, then wait for the corresponding interval of time to elapse before drawing the next frame.

While a movie is being presented, the user may interfere by asynchronously applying external interrupts through mouse commands in order to displace some element apart from its current position or to change some of its current attributes.

In order to apply such disturbances to the movie, the user simply drags an element through the window, or clicks first a function button, then the element to be affected. In these cases, interrupt the former operation, accept the disturbance as a command to be executed and interpret it by applying a corresponding handling routine, as follows:

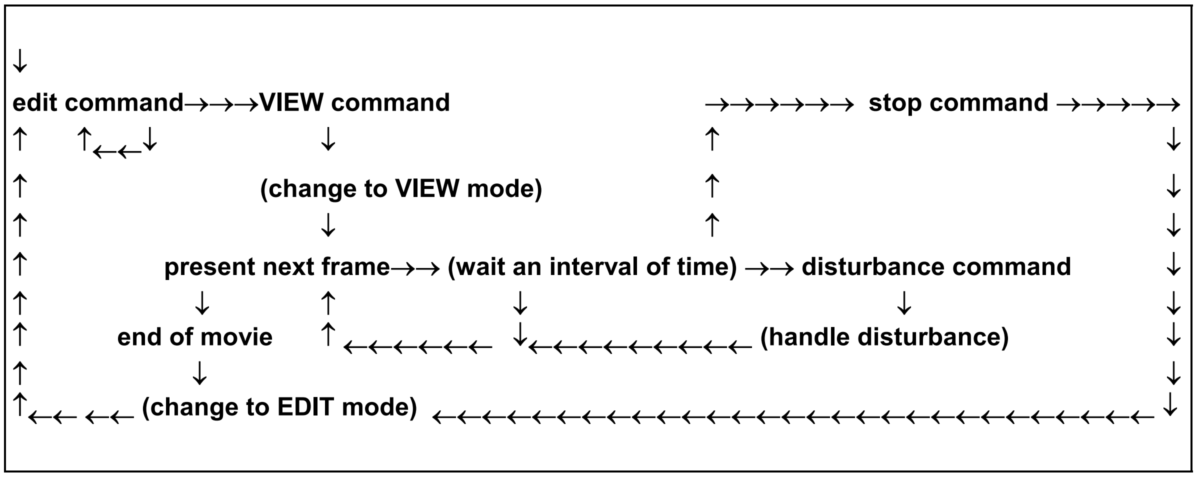
- *dragging* – interpret dragging as a command to displace the element to a new position, then interpolate a new straight path from the new position to the original final position of the element; erase the remaining path for that element and insert a new one; resume the interrupted routine.
- *changing color* – apply the requested color change to the element and resume.
- *drawing a color path* – calculate for the current remaining path of the chosen element the intermediate colors to be imposed, then correct the corresponding color attributes for the element in all remaining frames, then resume.
- *rotation* – split the requested rotation action into intermediate partial rotations, then apply them to the corresponding attribute for the chosen element in all remaining frames, then resume.
- *fading* – calculate the color path to be followed by the element in order to fade properly; correct the corresponding attribute on the remaining path of the element and resume.

Implementation Issues

Now we can draw a model for the implementation of our system from the specifications above. At a first implementation level, it may be viewed as a system that operates in two modes: in the first mode of operation, the system acts as an editor, and in the second mode, as a viewer. System starts in edit mode, and executes an arbitrary number of edit commands. Then the user issues a VIEW command, and watches the movie continuously, then he or she issues a stop-command in order to return to halt the movie and enter edit mode again. Edit mode is entered automatically at the end of the exhibition of the current movie. While the movie runs, the user is allowed to disturb it by issuing disturbance commands. The following expression defines a regular–language approximation to the command language of our system:

((edit-cmd)* (VIEW-cmd) (disturbance-cmd)* ((stop-cmd) | (end-of-movie-event)))*

The figure below depicts a simplified command flow and the changes in the mode of operation in our system, with no regard to asynchronous issues.



Edit commands may perform the following functions:

- create a blank frame by clicking a button
- clear current frame by clicking a button
- insert a new geometric shape into a frame by clicking some shape button, then clicking the frame at the desired coordinates and dragging the cursor until the shape acquires the desired aspect.
- change the position of an already existent shape by clicking the desired shape and dragging it to the new position in the frame

- change the size of an existent shape by selecting the shape, then dragging its corner until the shape acquires the wanted format.
- change other shape parameter – color, rotation angle, fading effects – by clicking the adequate function button, choosing the new parameter value, and then clicking the target shape.

The set of edit commands is expressed by the following expression:

```
(edit-cmd) = (new-frame-button) | (clear-button) |
             (shape-button) (click)(drag) |
             (select-shape)(drag) |
             (select-shape)(drag-corner) |
             (color-button)(color-select)(click) |
             (rotation-button)(angle-select)(click) |
             (fade-button)(fade-select)(click)
```

The set of disturbance commands is the same, except for the single-click commands:

```
(disturbance-cmd) = (select-shape)(drag) |
                    (select-shape)(drag-corner) |
                    (color-button)(color-select)(click) |
                    (rotation-button)(angle-select)(click) |
                    (fade-button)(fade-select)(click)
```

The former specifications, including the language definitions above, may be used as a general direction for the implementation of our system.

In [9] (elsewhere in these proceedings) we sketch an adaptive statechart representing our system, and we shortly describe an implementation prototype constructed following this model.

Conclusion

An extension of traditional statecharts is used, which imports the concept of adaptability from adaptive automata, resulting a formalism that may be used as a practical approach to support more efficient implementations than similar models proposed in the literature.

In this article we discuss adaptive formal models as tools for expressing in a more natural way the behavior of self-modifying systems.

We illustrate the concept of adaptive methods by applying it to the specification and implementation of a simple system used for creating and viewing interactive compressed 2-D movies with moving geometric shapes that allow operator interference at run-time.

Dynamic modifications are applied to the elements of the set of moving shapes present in the movie frames in run time.

Adaptive features are explored in the system to express the dynamic behavior of the interactive movie, illustrating the power of this feature.

Besides allowing direct use of theoretical formulation in order to produce good implementations, adaptive techniques allow knowledge acquisition to be included in the resulting models.

This effect is achieved through self-modification of their own reactions to external stimuli in response to new information acquired from input data.

Therefore, intelligent systems may be represented through adaptive models. In particular, the graphical nature of adaptive statecharts make them better to understand and to design than equivalent clause-based or algebraic formulations.

Usual models are static, and some eventual dynamic behavior of the system being modeled must be considered separately. When representing dynamic reactive systems, our model allows the whole system to be modeled at once.

Adaptive models have many applications, and the one presented in this paper is a very simple and informal case study, understandable even by non-specialists, in which the underlying concepts of adaptive formalisms and of reactive system modeling may be identified.

References

[1] José Neto, J. Adaptive Automata for Context-Dependent Languages *ACM SIGPLAN Notices*, Vol. 29, n. 9, September 1994, p. 115-124

[2] Cabasino, S.; Paolucci, P. S.; Todesco, G. M. Dynamic Parsers and Evolving Grammars *ACM SIGPLAN Notices*, Vol. 27, No. 11, 39-48 (November 1992).

[3] Almeida Jr., J. R. STAD - Uma Ferramenta para Representação e Simulação de Sistemas. Através de Statecharts Adaptativos *Tese de doutoramento – Escola Politécnica da Universidade de São Paulo* – São Paulo, Brasil, 1995 (in Portuguese).

[4] Pereira, J.C.D.; José Neto, J. Um Ambiente de Desenvolvimento de Reconhecedores Sintáticos Baseado em Autômatos Adaptativos. *Simpósio Brasileiro de Linguagens de Programação*, 1997 (in Portuguese).

[5] Shutt, J.N. Recursive Adaptable Grammars. *Ph. D. Thesis, Worcester Polytechnic Institute*, 1993.

[6] Christiansen, H. A survey of adaptable grammars *.SIGPLAN Notices*, vol.25, n.11, p.35-44, 1990.

[7] Burshtein, B. Generation and recognition of formal languages by modifiable grammars. *ACM SIGPLAN Notices*, v.25, n.12, p.45-53, 1990.

[8] José Neto, J. , Almeida Jr. , J. R. , Santos, J. M. N. Synchronized statecharts for reactive systems. *IASTED'98 - Applied Modelling and Simulation* Honolulu, Hawaii, USA Aug. 12-14. 1998, p. 246-251

[9] Almeida Jr., J.R., José Neto, J. Using adaptive models for system description. *IASTED'99 - Applied Modelling and Simulation* Cairns, Australia Sept. 1-3. 1999