SYNCHRONIZED STATECHARTS FOR REACTIVE SYSTEMS

JOÃO JOSÉ NETO, JORGE RADY DE ALMEIDA JÚNIOR, JOSÉ MARIA NOVAES DOS SANTOS
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 , novaesjm@usp.br

ABSTRACT

When modelling reactive systems, one has to adequately choose tools and notations that are consistent and appropriate to comfortably express all relevant features of a system. There are many ways and notations to describe the structure and behavior of systems. Graphical notations are uaually attracting for being intuitive and readable. Statecharts and Petri Nets constitute such a kind of notation, although being different in their conceptual level. The present paper proposes a hybrid notation based on these two ones, to be used in a tool for describing reactive systems, and discusses some aspects of such class of notations. Hybrid approaches let their users to exploit the best features from each composing notation, and may lead to better and more readable representations, provided their users are well-disciplined. A computer program supporting that notation is also reported, and a simplified real-world example is used to illustrate the use of this tool.

Keywords: statecharts, reactive systems, synchronisation, modelling

INTRODUCTION

Statecharts, as introduced by Harel [1], make an excellent graphical notation for describing reactive systems. *Petri Nets* [2] are also used as a good tool for analysis and design of synchronised processes. While statecharts, as a higher-level notation, tend to hide low-level temporal details of the described phenomena, with Petri Nets the representation of functional aspects of the process is not immediate.

In many situations, it would be adequate to use some well-known notation in order to represent a process, in its structural and functional aspects, without leaving apart the details of its temporal aspects. In such a case, hybrid notations seem to be adequate.

Many variants exist both for Statecharts and Petri Nets. In the implementation mentioned in this work, only a restricted kind of Petri Nets is considered for working along with an adaptive variant of Harel's Statecharts.

Adaptive Statecharts [3] are extensions that increments the features of traditional Statecharts with the so-called adaptive actions. Adaptive actions are driven by special transitions of the Statechart, and they allow the current Statechart to modify itself, by altering its topology and, consequently, changing its functionality, as a response to external stimuli.

Although the main purpose of the present paper is not studying adaptive aspects of the Statecharts, the adaptive feature is mentioned here because it has been implemented this way in the tool we have developed to support this hybrid notation [4].

SYNCHRONISED STATECHARTS

In this paper we report a proposal of a hybrid model in which we integrate Petri Nets to Adaptive Statecharts. With such a compound device it has become easier to use both notations in one only environment, allowing one at the same time to explicitly take care of low-level synchronising aspects of the reactive system under appreciation (by means of Petri-Nets connecting Statecharts) while describing all other elements of that system at a higher level (by means of their Statechart descriptions where eventual dynamically-changing behaviour aspects may be expressed through adaptive actions).

Using such kind of hybrid modelling devices show both advantages and disadvantages. The main disadvantages they bring refer to:

a) In order to fully explore the features of any hybrid notation, users need to have good familiarity with more than one notation.

b) The non-uniformity in the level of detail each notation demands for. In the present case, there is a level gap between high-level statecharts and low-level Petri Nets.

c) Building tools that give support to hybrid models demand for the implementation of all models the hybrid one is made up from, so inheriting complexity from all of them.

d) As long as each already-existent notation is self-contained in many aspects, it is always possible to fully describe the system being modelled without importing any extra feature from other notations

e) By often providing alternative ways to express the same phenomenon, no enforcing of uniformity is imposed by the notation, demanding from their users a higher degree of discipline than in the case of non-hybrid notations in order to achieve readable and expressive descriptions.

The main advantages of using hybrid modelling lie on the following:

a) Hybrid notations allow one to employ the best features of each composing notation, so giving their users a larger repertoire of tools, and allowing them to choose the most expressive ones

b) Petri Nets are particularly suitable for explicitly representing temporal aspects of a system, complementing the adequacy of Statecharts to represent hierarchical, structural and logical aspects of the system.

c) Already-made Statecharts may be used almost directly in the hybrid model without needing any internal modification, so, in general, composing a number of Statecharts with Petri Nets does not require opening them, provided that proper interfaces have been adequately established.

d) Petri Nets are devices that have been extensively studied, for which many useful analysis results are available from theory, both in the form of theorems and algorithms.

e) By means of a hybrid model supporting tool, non-familiar users may incrementally learn each component of the hybrid model, as well as the whole hybrid model.

f) Users may choose among the tool's features in order to specialise it in favour of each particular problem's needs.

g) Such a tool may be used as a pedagogical device, allowing their users to be hierarchically trained in increasingly-difficult issues of the various theoretical and practical aspects of the models and of the tool itself.

A WORKING ENVIRONMENT FOR SYNCHRONISED STATECHARTS

STAD-S is a tool that has been developed with the intent of providing an environment for experimenting and testing synchronisation issues in reactive systems described by models based on (adaptive) statecharts.

Adaptive Statecharts are extensions of Harel's statecharts that allow their users to attach, to selected transitions, calls to potentially parametric adaptive functions (*adaptive actions*).

At the time those optional adaptive transitions are performed, the corresponding attached adaptive action take place by applying editing operations to the adaptive statechart itself.

Three basic kinds of elementary actions are allowed in adaptive functions in order to express the specification of the desired statechart modifications:

- *inspection actions* locate transitions in a statechart that match a given template
- *removal actions* allow eliminating transitions matching some template
- *insertion actions* allow adding bubbles and transitions to the statechart

As we have mentioned before, the present paper will not deal with the adaptive features of the notation, but we refer to them here only for sake of completeness. For further information on this subject see [5].

Systems whose data have been included in a STAD-S-developed model are called *projects*, which may be themselves hierarchically split into *subprojects*, in an arbitrary depth of levels of detail. All data referring to registered projects are hold in a database by STAD-S.

A graphical editor is available for users to specify their statecharts, allowing bubbles and transitions to be easily created and specified. At its creation time, a bubble may be tagged as being expandable, allowing the creation of arbitrarily hierarchical specifications.

Transitions from bubble to bubble are triggered by either independent or dependent events. The former refer to externally-generated events, which cause initial transitions to occur, while the later ones represent events generated internally to the currently active statechart, and are responsible by the occurrence of intermediate transitions.

In our tool, transitions driven by independent events are represented in dashed lines, while those related to dependent events are drawn with solid lines.

Bubbles are connected through directed links (direction is represented by an arrow), made up by the following elements:

- one *driving event*, which allows the transition to occur provided that its associated condition holds. The event is the only mandatory component of a link.
- one *condition*, under which the associated event can activate its corresponding transition. This condition indicates whether or not the bubble under study is currently active. Conditions are optional components of the links between bubbles. When omitted, the corresponding transition activation will occur unconditionally.
- a *trigger*, indicating which eventual other link is to be activated as a response to the execution of the current transition. This is also an optional element, and its omission indicates that no other link is to be activated.
- an optional pair of numbers, indicating a *delay* and the minimum value for the *response time* specified for the transition.
- an optional *adaptive action*, represented by a call to an adaptive function, which may modify the statechart in order to change its behaviour if necessary.

STAD-S checks for consistencies whenever a statechart is detailed one further level, and both informations from the preceding and the succeeding levels are verified. Inputs and outputs to each bubble are tested so that in both neighbour hierarchical levels all of these elements must be also present, so enforcing adequate connectivity.

Inputting adaptive functions to STAD-S is done in a very similar way, by using the same editor. Whenever an adaptive function is called by some link, it is instantiated as an adaptive action associated to that link, for being executed whenever the corresponding transition occurs.

Adaptive actions are made up from basic editing operations to the statechart. They are used for specifying the insertion of new bubbles and links, and the deletion of existing links.

Sources and destinations of links, bubble names, names of events associated to links, and the identification of a link to be excluded are the elements allowed as parameters by an adaptive function.

At the time an adaptive function call is attached to a link, it instantiates an adaptive action, which will, at its run time, extract values from the arguments, assigning them to their corresponding formal parameters.

STAD-S allows one to simulate the execution of a given statechart. Once completely specified, an (adaptive) statechart may be simulated, step-by-step or automatically, by the tool, provided the user properly inserts the needed information about independent events.

By convention, its graphical user interface indicate to the user all currently active bubbles by redrawing them with a thicker contour. The same is done to the lines representing currently working links. In automatic simulation mode, this convention give the user a way to follow the activity of the system through an animated picture of the simulation process.

All already mentioned features have been implemented in a former tool, named STAD. STAD-S was built as an extension to STAD, allowing the specification of compound systems built up from a set of (adaptive) statecharts, linked through Restricted Petri Net connections.

In the current implementation of STAD-S, severe restrictions are imposed to the shape of linking nets, in order to keep it simple to simulate phenomena related to the synchronisation of the statecharts.

An elementary Restricted Petri Net connection is limited to have m inputs leading to m corresponding places through simple transitions, and n outputs coming out of transitions fed by unique places. The m input statechart bubbles reach a single transition, from which the n remaining bubbles are directly reached (see fig. 1). All those bubbles are mirrowed by STAD-S as Petri-Net places.

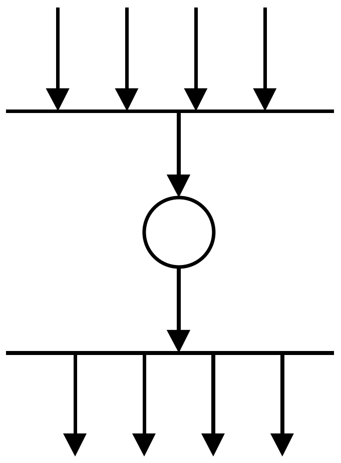

Fig. 1 - General shape of a Restricted Petri Net used to interconnect statecharts

However, the imposed restrictions do not conceptually restrict the kind of synchronisation to be used, because it is always possible to artificially connect arbitrary basic Restricted Petri Nets through dummy statecharts, acting as mere places, until an equivalent form of the desired Petri Net shape is obtained.

As we mentioned before, using this hybrid notation allow defining systems which are composed from already available (adaptive) statecharts, with only minor changes.

Different components are independently represented, and eventual interactions and temporal dependencies among them are explicitly denoted by means of Restricted Petri Net connections.

STAD-S program consists of two large functional modules: one is used to specify (adaptive) statecharts, performing graphical edition of statecharts in a fashion similar to Harel's definition, and the second one complements the hybrid notation by allowing the input of connections among already specified statecharts through Restricted Petri Nets.

In these connections, each Restricted Petri Net is identified by:

- a unique name, used for identification purposes
- a list of the input statechart bubbles to be connected through the Restricted Petri Net
- a list of the output statechart bubbles to be activated in response to a token generated as output by the Restricted Petri Net

Restricted Petri Nets reflect the kind of connections allowed in the present implementation of STAD-S, but a future version is planned to accept and simulate more general shapes of Petri Nets, and to run a set of consistency tests on the models.

ILLUSTRATING EXAMPLE

The following illustrating example uses the proposed hybrid notation to describe a system composed by rather independent subsystems, each one represented by a non-adaptive statechart, interconnected by some kind of protocol, whose synchronisation is explicitly represented by Restricted Petri Nets, interconnecting selected bubbles inside the statecharts.

This example refers to a system in which a diversity of customers ask several servers for different services. When a service is ordered, the system checks for the rights of the asking customer, and rejects requests not allowed for that customer. Otherwise, an adequate server is requested to provide the service. Anyway, all involved modules are notified.

Although being a very short example, it represents a somewhat complete case for study, since it deals with closed feedback systems, sketching a nice working solution to a wide class of usual practical problems, in fields such as insurance, hotel and ticket reservations, schools, services, renting of goods, process control and supervision and many others.

In the example below, we show one only level of detail for the statecharts, in order to identify the set of transitions affected by synchronisation, so most of their logic is omitted for it is irrelevant to our discussion.

Additionally, we do not replace by Restricted Petri Nets all those transitions from the set of statecharts. Although that would apply to all synchronising transitions in the set of statecharts, we kept them all but one, randomly chosen only to illustrate that this transformation may be applied strictly over synchronising transitions that are of interest.

In order to present a concrete practical case study we instantiate our general example by applying it to represent the simplified behaviour of a Health Insurance company.

In figure 2 a model is proposed, based on statecharts only. Dashed lines have been superimposed to the figure in order to show existing (implicit) communications among statecharts. The system under study is made up by four independent subsystems, each one represented by a separate statechart: customer, company, plan and resource, drawn following Harel's conventions [1].

Statechart customer represents people who had acquired some Health Plan from a company modelled by statechart company. Statechart resource is a rough representation of all hospitals, laboratories, physicians and other elements available. Statechart plan represents the health plan we are talking about.

When a customer requests some service (for example, a clinical test), the company controlling his insurance checks whether or not the requested service is covered by the given plan. In affirmative case, it enables the request, otherwise the request is rejected.

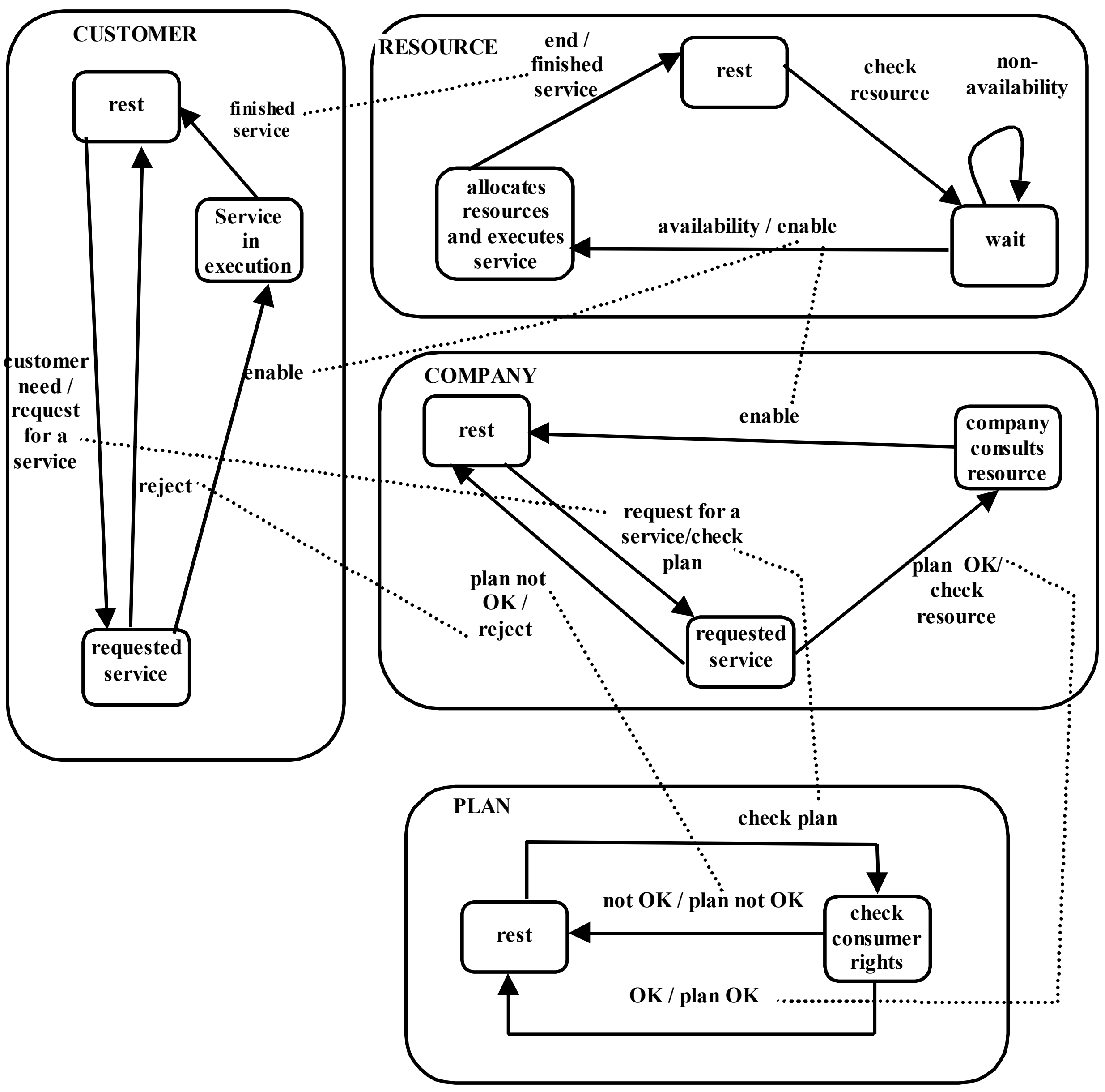

Fig. 2 – Illustrating example – health plan

The accomplishment of the requested service will be obviously conditioned to the readiness of the corresponding needed resources (in our case, proper laboratory resources must be available to perform the requested test).

Supposing that our only interest is about details on the synchronisation issues related to the event **finished service**, we will change the connections labelled with that event in the two involved statecharts (customer and resource) into an equivalent connection made up from Restricted Petri Nets and a dummy connecting statechart.

Figure 3 illustrates the substitution of the transitions with attached event **finished service**, communicating and synchronising statecharts customer and resource. The corresponding transitions are replaced by two Restricted Petri Nets RPN1 and RPN2, and a dummy statechart has been included to accommodate the notation accepted in the present version of STAD-S.

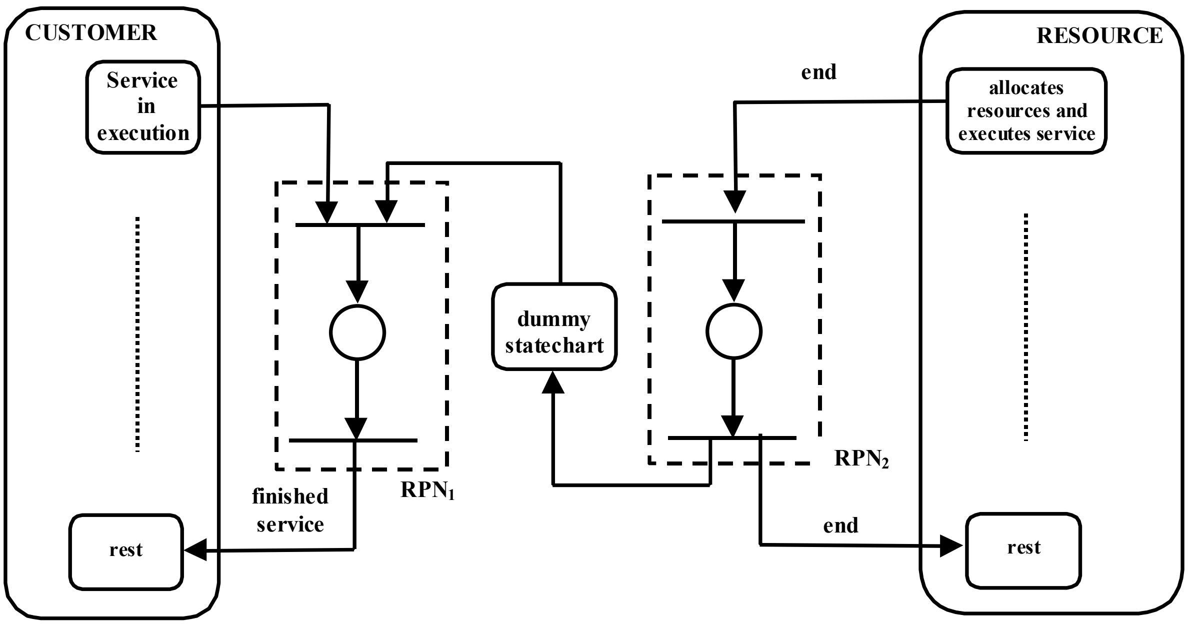

Fig. 3 - Equivalent Petri Net connection for event **finished service**

CONCLUSIONS

Hybrid notations are not a must, but they can ease viewing and analysing directly some details that would be harder to handled otherwise.

As we can see from the example above, in real problems there are usually so many elements in the model that it often very difficult even to locate in the set of statecharts all elements we are interested in, especially because these dependencies are not explicitly drown in the graph.

By performing the transformation suggested in this article, an equivalent graph is obtained, where all synchronising features we want to study are explicitly shown.

With the help of our tool, these elements may be stimulated adequately and the results of their simulation may be observed, giving the user further insights on their synchronisation problems.

Obviously it is not wise to blindly apply this transformation to all synchronising transitions, because that would convert all statecharts into Petri Nets, so losing the advantages of their higher-level expressiveness.

STAD-S has also adaptive features, not covered in this paper, but useful when modelling systems with input-driven dynamic behaviour or some kind of learning capabilities.

Finally, STAD-S is a tool with high pedagogical potential, that may be used in several situations as a laboratory or as a teaching aid, through which many issues may be taught and learned: reactive systems, statecharts, Petri Nets, learning devices, synchronisation, communication, simulation and many others, all very important in computer science and engineering.

REFERENCES

[1] HAREL, D. Statecharts: a visual formalism for complex systems. Science of Computer Programming, v.8, n.3, p.231- 74, Aug. 1987b.

[2] PETERSON, J. L. Petri nets. Computer, v.9, n.3, p.224-252, Sept. 1997.

[3] ALMEIDA JUNIOR, J. R. STAD - Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos, São Paulo, 1995, 202p. Doctoral Thesis (in Portuguese) - Escola Politécnica, Universidade de São Paulo.

[4] SANTOS, J. M. N.. Um formalismo adaptativo com mecanismo de sincronização para aplicações concorrentes, São Paulo, 1997, 98p. Thesis (in Portuguese) - Escola Politécnica, Universidade de São Paulo.

[5] JOSÉ NETO, J. Adaptive automata for context-sensitive languages. ACM Sigplan Notices, v.29, n.9, p.115-24, Sept. 1994