Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 115 (2005) 49–57
www.elsevier.com/locate/entcs

Interacting Extended State Diagrams

Gwen Salaün[a,1,3] and Pascal Poizat[b,2]

[a] *DIS, Università di Roma "La Sapienza", Roma, Italy*

[b] *LaMI, Université d'Évry Val d'Essonne, Évry, France*

Abstract

Integrated formal description techniques are a promising approach for the specification of multi-aspect systems. In this context, we have proposed a formalism, called Extended State Diagrams (ESD), combining in an homogeneous framework state diagrams and formal data description languages. Our purpose is here to enhance the implicit ESD communication mechanisms with explicit descriptions of communications, which is achieved using synchronization vectors. The use of interaction diagrams is also discussed.

Keywords: Formal Specification Integration, State Diagram, Formal Datatype, Synchronization Vector, Interaction Diagram.

1 Introduction

In the last few years, the increase of systems complexity made it necessary to describe them under different aspects. This separation of concerns appeared at both the programming (the so-called Aspect-Oriented Programming) and the Specification and Design (AOSD) levels. The use and henceforth the definition of integrated formal description techniques is a promising approach for the specification of such multi-aspect systems. Following several experiments we made on this subject [1], we have proposed an integrated formalism,

[1] Email:salaun@dis.uniroma1.it
[2] Email:poizat@lami.univ-evry.fr
[3] This work is partially supported by Project ASTRO funded by the Italian Ministry for Research under the FIRB framework (funds for basic research).

doi:10.1016/j.entcs.2004.09.028

Extended State Diagrams [4] (ESD). ESD uses formal and graphical notations with the objective of taking advantages of both: user-friendliness and readability of graphical notations such as the UML, the de-facto standard for software modelling, and formal notations to enable verification and animating mechanisms [5]. The static and functional aspects of systems are described using formal data description languages such as algebraic specifications, Z or B. The dynamic aspects of systems (events, behaviours and communications) are described using extended state diagrams dealing with the formal datatypes of the static aspects.

The UML notation is used in ESD to denote event sendings (`D2^tick(n+1)` in diagram `D3`, Fig. 1, for example), and the ESD integration semantics then builds on this to propose a flexible mechanism of communication constraints to express possible communication models. These constraints expressiveness has been illustrated in [4] to deal with the usual object-oriented asynchronous message-passing communication. However, this technical choice is not an ideal one. First, it needs the communication constraint to be redefined each time the specifier needs a specific communication model. Furthermore, the lack of means to make the communication model between ESDs explicit is error-prone and harms reuse of specification modules as communication constraints are to be found into sequential components.

To improve the means of interaction between ESDs, we extend here our initial proposal with alternative and complementary communication mechanisms, still preserving the formal consistency of the whole integrated formalism: synchronization vectors. The use of interaction diagrams is also discussed. Both plug in a simple way onto the ESD semantics we gave in [4]. Accordingly, the specification of interactions may be undertaken straightforwardly (no need for additional formalisations).

This article is structured as follows. In Section 2, we summarize the ESD approach. Sections 3 and 4 then deal with our extension proposals. Finally, we end with concluding remarks on this work.

2 Extended State Diagrams

This section gives the necessary insights into the ESD semantics. A more comprehensive presentation of our approach may be found in [4].

Syntax. ESD extends state diagrams (such as Statecharts or the UML ones) with:

- *data boxes* in which formal datatype modules are imported and formal variables are declared and typed

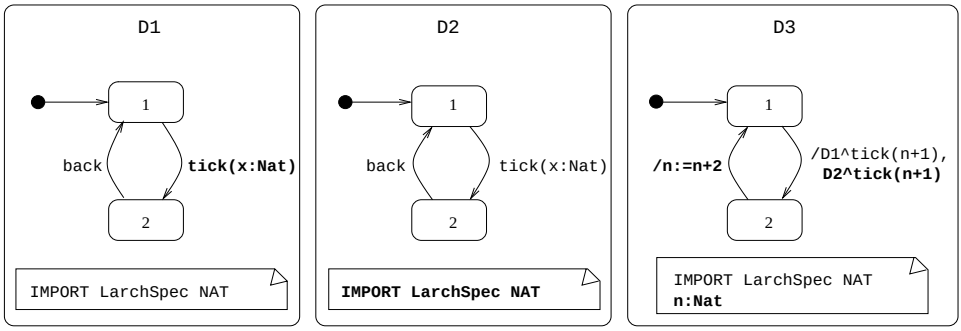

Fig. 1. Three interacting ESDs

- in transitions, the use of formal datatypes (algebraic terms, Z or B operations) in guards, in receptions, and in the action part (both in actions and in event sendings).

This is illustrated in Figure 1 on a simple example of three interacting diagrams. From the left hand side to the right hand side, bold labels respectively denote: (`D1`) the reception of a natural value in a variable `x` through the `tick` event, (`D2`) the import of the data description module `NAT` which is written using the Larch specification language, (`D3`) the modification of value of the `n` variable in the action part of a transition as well as its definition in the data box, and the sending to `D2` of value `n+1` through the `tick` event.

Semantics. Let us first note that our goal is not to give a formal semantics to some specific dialect of state diagrams (a lot do exist, see [4] for references). We rather aim at formalising the extension of such diagrams with formal datatypes, in a modular way, that is extending their semantics. We take as an hypothesis that these semantics (called *basic* hereafter) are given in an operational way, using labelled transition systems (LTS).

Our semantics is given using groups of rules [4]. The first two groups (R1 and R2) are used to type-check terms and give them an interpretation using evaluation mechanisms. The R3 group is used to extend the basic semantics to describe the individual evolutions of extended diagrams, which is achieved by extending the states of the basic semantics with variable bindings (denoted by $\mathcal{E}$), an input event queue (Q_{in}, where events arrive and are stored) and an output event queue (Q_{out}, denoting sent events). This extension takes possible value receptions and actions into account. This yields a set $\{D_i : (\underline{S}, \underline{S_0}, \underline{T})\}$ of LTSs. The R4 group is then used to describe possible modifications in the queues of each diagram resulting of an open-system semantics (additions in input queues, removals from output ones). This yields again a set $\{D_i : (\underline{S}^{\text{open}}, \underline{S_0}^{\text{open}}, \underline{T}^{\text{open}})\}$ of LTSs where, for each D_i, $\underline{T}^{\text{open}} \subseteq \underline{T} \times E_{\text{in}}^* \times E_{\text{out}}^*$, with E_{in} (resp. E_{out}) being the set of all input events (resp. output events) of D_i. As

usual with LTS, $(s, label, s', E^+, E^-) \in \underline{T}^{\text{open}}$ is denoted by $s \overset{label}{\rightarrow}_{E^+,E^-} s'$. Rule R5, given a communication model constraint CC, then builds a global LTS $(\overline{S}, \overline{S_0}, \overline{T})$ resulting from the product of the R4 ones, that is $\overline{S} = \Pi_{D_i} \underline{S}^{\text{open}}(D_i)$, $\overline{S_0} = \Pi_{D_i} \underline{S_0}^{\text{open}}(D_i)$, and $\overline{T} = \{\overline{t} \in \Pi_{D_i} \underline{T}^{\text{open}}(D_i) \mid CC(\overline{t})\}$.

CC may be defined for example to relate legal open-systems queue modifications between two diagrams in an asynchronous message-passing communication model:

$$CC(s_1 \overset{l_1}{\rightarrow}_{E_1^+,E_1^-} s'_1, \ \ldots\ , s_n \overset{l_n}{\rightarrow}_{E_n^+,E_n^-} s'_n) \Leftrightarrow$$
$$\forall \text{sender} \in 1..n,\ \forall D_{\text{receiver}} \hat{}\, e \in E^-_{\text{sender}},\ D_{\text{receiver}} \in \cup_{i \in 1..n} D_i \Rightarrow e \in E^+_{\text{receiver}}$$

This separation of (sets of) rules, enables one to reuse specific ones and replace or specialise other ones to deal with specific needs. It is this modularity which enables one to propose alternatives for the sets of rules mentioned above, and as far as this article is concerned to enhance R5.

3 Interacting ESDs with Synchronization Vectors

An alternative to implicit communication policies into component is to specify interactions using synchronization vectors, which are expressive, formal and readable (even if only textual) means to write interactions between ESDs. Synchronization vectors have originally been introduced by Arnold and Nivat [2] and are used for instance in the AltaRica formalism [3]. An extension of this initial concept is defined in Korrigan [7] in which advanced synchronization vectors using temporal logic formulas enable one to compose automata thanks to different strategies.

Syntax. If technically we use synchronization vectors, our goal is to keep a good level of readability, hence finding an intermediate level between [2,3] basic vectors (which model only basic synchronous communication) and Korrigan more expressive, but also much more difficult to use, temporal formulas. Our vectors can be used to model different synchronization policies between ESDs, as for exemple:

- asynchronous, oriented, one to n, different names: $[a!,\varepsilon,b?,\varepsilon,c?]$
- synchronous, value passing, one with many: $<a!,\varepsilon,b?,\varepsilon,c?>$
- synchronous, value agreement, many with many: $<a!,\varepsilon,b!,\varepsilon,c!>$
- synchronous, value negotiation, many with many: $<a?,\varepsilon,b?,\varepsilon,c?>$

$< \ldots >$ vectors deal with synchronous communication, whereas $[\ldots]$ ones deal with asynchronous communication. Prefixing events with diagram iden-

tifiers (*e.g.* <*D1.a!,D2.b?,D3.c?*>) is not needed. Correspondence is obtained through the order of events in vectors. In vectors the only information kept on events is whether they are input (*e.g.* e!) or output (*e.g.* e?) events. The information on variables or values bound to the events is dealt with by rule R3 of the ESD semantics. ε is used as usual with synchronization vectors to express that the corresponding ESD does nothing (this will match its "stuttering-steps" ε transitions) while other synchronize. Asynchronous vectors contain exactly one output event.

Semantics. The formalisation of interaction through synchronization vectors is made in such a way that its rules may replace the R5 one in our ESD semantics. First of all, remember that a global transition $\overline{t} = (s_1 \xrightarrow{l_1}_{E_1^+,E_1^-} s'_1, \ldots, s_n \xrightarrow{l_n}_{E_n^+,E_n^-} s'_n)$ is only valid (R5) if CC was true for it. Here CC is parameterised with a set V of vectors. A transition is valid for a set of vectors if it is valid for one: $CC(\overline{t}, V) \Leftrightarrow \exists v \in V,\ \overline{t} \models v$.

Let *type* be the function defined such as for any input event $e?$ in vectors, $\text{type}(e?) = E_{\text{in}}$, and for any output event $e!$, $\text{type}(e!) = E_{\text{out}}$. Moreover, $\text{type}(\varepsilon) = E_\varepsilon$. The rule for synchronous communications is:

$$(s_1 \xrightarrow{l_1}_{E_1^+,E_1^-} s'_1, \ldots, s_n \xrightarrow{l_n}_{E_n^+,E_n^-} s'_n) \models < e_1, \ldots, e_n > \Leftrightarrow$$

$$\exists \overline{v},\ \forall i \in 1..n \begin{cases} l_i = e_i(\overline{v}) & \text{if type}(e_i) = E_{\text{in}} \quad (a) \\ Q_{\text{out}}(s_i) - Q_{\text{out}}(s'_i) = \{e_i(\overline{v})\} & \text{if type}(e_i) = E_{\text{out}} \quad (b) \\ l_i = \varepsilon \wedge Q_{\text{out}}(s_i) = Q_{\text{out}}(s'_i) & \text{if type}(e_i) = E_\varepsilon \quad (c) \end{cases}$$

This rule deals at the same time with event name and types which have to match between the vector and the transitions, and with value passing. Input events and ε are in the transition labels of the ESD LTSs. Output events are not, they are put into the output queues of senders (R3 rules) and extracted when communicating (R4 rule). Hence, they correspond to differences, in-between source and target states of transitions, in the corresponding ESD output queue, Q_{out}. All output events have to agree on the same value ($\overline{v}$), while input events will receive it. The rule presented here deals with single value passing but may be generalized to n values as in LOTOS [6].

Asynchronous communication uses the E^+ and E^- elements of global transitions. The rule for asynchronous communication is then obtained replacing (a) with $e_i(\overline{v}) \in E_i^+$ and (b) with $e_i(\overline{v}) \in E_i^-$.

Less strict versions of synchronization may be expressed with less strict versions of the vectors. For example, broadcast communication (*i.e.* only

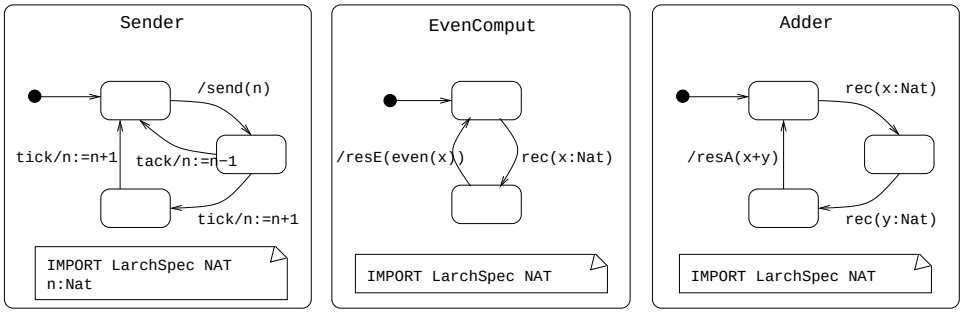

Fig. 2. Three interacting ESDs without implicit communication

some of the required receivers may really synchronize, as opposed to multicast communication where all registered receivers have to synchronize) may be expressed.

Application. Our example (Fig. 2) is made up of three interacting diagrams: a sender which transmits naturals, an even computer which tests if numbers are even, and an adder which adds two naturals. A value is broadcasted synchronously by the sender to the even computer, then another value is sent to the adder. Finally, the result of the addition is communicated to the even computer. The vectors corresponding to the communications for this example, with ESD reference order being `Sender`, `EvenComput`, `Adder`, are:

$$\{< \texttt{send!}, \texttt{rec?}, \texttt{rec?} >,\ < \texttt{send!}, \varepsilon, \texttt{rec?} >,\ < \varepsilon, \texttt{rec?}, \texttt{resA!} >\}$$

Note that these vectors do not imply any order in the interaction firing, only legal synchronizations. This drawback of synchronization vectors may be lifted using interaction diagrams.

To end this section, let us stress that xCLAP [5] makes it possible to compute synchronous products (with our transition-related vectors but also state-related ones), a skill recovered from its predecessor, CLAP [8]. These expressive means could also be applied to ESDs and the whole specification (some extended state diagrams and synchronization vectors) could be animated using the xCLAP simulation features.

4 Interacting ESDs with Interaction Diagrams

Our goal in this section is to give insights that ESD interactions could also be described using interaction diagrams (this term taking into account their dialect such as MSC or UML sequence diagrams).

We describe in Figure 3 the possible interactions between our three example components (Fig. 2). As MSC do not enable components to synchronize

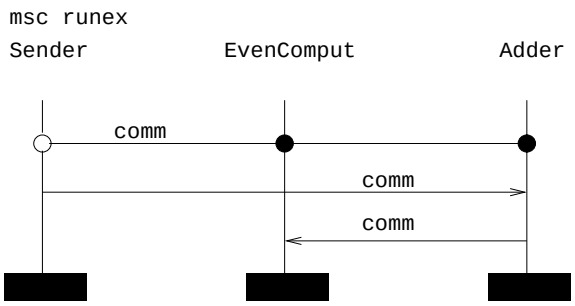

Fig. 3. MSC interaction for the three ESDs

$$\frac{\begin{array}{c} S = \{S_{D_1}, ..., S_{D_n}\} \\ S_{D_1} S_{D_2} \ \triangleright \ e \\ S_{D_1} \xrightarrow{e} S'_{D_1} \quad S_{D_2} \xrightarrow{e} S'_{D_2} \end{array}}{\mathrm{M} \ \models \ S \xrightarrow{apply(e)} S[S'_{D_1}/S_{D_1}, S'_{D_2}/S_{D_2}]}$$

Fig. 4. MSC interaction semantics

on events with different names (which is a drawback as this harms the reusability levels of components), Figure 2 ESDs synchronized transitions should be renamed in `comm`. However, temporal ordering of synchronizations may be expressed (which was not the case with synchronization vectors).

Semantics. MSCs yield different CC rules dealing with the different kinds of interactions they define. Taking for example an interaction between any two diagrams D_1 and D_2 on event e (such as the one between `Sender` and `Adder` on `comm`), the CC rule is: $CC((S_{D_i} \xrightarrow{l_i} S'_{D_i})_{i \in 1..n}, \mathrm{M}) \Leftrightarrow \mathrm{M} \ \models \ \{(S_{D_i})_{i \in 1..n}\} \xrightarrow{apply(e)} \{(S_{D_i})_{i \in 1..n}\}[(S'_{D_i}/S_{D_i})_{i \in 1..2}]$, with notation $E[x/y]$ denoting as usual the set E' which is equal to E substituting x for y. CC uses transitions labelled with *apply* obtained from deduction rules. The rule here is described in Figure 4. As for state diagrams [4], we do not aim at formalising interaction diagrams, but rather being able to reuse their different semantics in a generic way in the definitions of our communication constraints semantics rules. $\triangleright$ is assumed to be the function of the chosen MSC semantics that yields legal transitions of the MSC models. Here it states that an interaction on e is possible between D_1 and D_2 which are respectively in states S_{D_1} and S_{D_2}.

Interaction diagrams are complementary with synchronization vectors as far as graphical representations and temporal ordering of the communication scheme are concerned. However, a shortcoming of the different interaction di-

agram semantics is that none of them deals with all the different types of communication we dealt with using synchronization vectors. Sequence diagrams proposed in UML 2.0 are much more satisfactory as far as expressiveness is the issue, but they are not (yet) fully formalised.

5 Concluding Remarks

Explicit communication policies increase the reusability level of integrated specification as no specific communication information is present in the sequential components. In this paper we have proposed extension mechanisms to explicitly specify the interactions between ESDs. Our synchronization vectors may express very different synchronization policies. The use of interaction diagrams is less expressive but takes benefit from its specifier-friendly graphical representation.

Related works include first process algebras (mainly LOTOS [6] as far are integrated specifications are concerned). However, their implicit synchronization policy is often hard to understand (for example the implicit synchronization on the ending of processes, δ). Moreover, as our synchronization vectors plug above our existing ESD semantics, we may express both synchronous and asynchronous communication, or the synchronization of n processes among m, which is not the case for LOTOS. An approach closer to ours are ISM [9]. ISM are high-level Input/Output Automata with the same expressiveness than ESD but a significantly improved structuring. However, the communication between ISMs is more restricted than ours because only asynchronous buffered communication is available.

Perspectives of this work are twofold. A first direction is to generalise our approach to a framework supporting general (and generic) means to specify interactions between dynamic specification modules. This would in some sense require the definition of a more general coordination language for a vast class of integrated specification languages. Our second perspective is to extend the xCLAP toolkit [5] animation and verification means using the ideas developed here (xCLAP itself is an extension of the CLAP [8] tool which dealt with simpler specification languages).

References

[1] Allemand, M., C. Attiogbé, P. Poizat, J.-C. Royer and G. Salaün, *SHE'S Project: a Report of Joint Works on the Integration of Formal Specification Techniques*, in: *Workshop on Integration of Specification Techniques with Applications in Engineering*, 2002, pp. 29–36.

[2] Arnold, A., "Finite Transition Systems," Prentice-Hall, 1994.

[3] Arnold, A., G. Point, A. Griffault and A. Rauzy, *The AltaRica Formalism for Describing Concurrent Systems*, Fundamenta Informatica **40** (1999), pp. 109–124.

[4] Attiogbé, C., P. Poizat and G. Salaün, *Integration of Formal Datatypes within State Diagrams*, in: M. Pezzè, editor, *International Conference on Fundamental Approaches to Software Engineering*, LNCS **2621** (2003), pp. 341–355.

[5] Auverlot, A., C. Cailler, M. Coriton, V. Gruet and M. Noël, *xCLAP: Animation of State Diagrams with Formal Data, Master's Degree Project, University of Nantes. Available at* http://www.dis.uniroma1.it/£\sim£salaun/xCLAP/ (2003), directed by C. Attiogbé and G. Salaün.

[6] Bolognesi, T. and E. Brinksma, *Introduction to the ISO Specification Language LOTOS*, in: P. H. J. van Eijk, C. A. Vissers and M. Diaz, editors, *The Formal Description Technique LOTOS*, Elsevier Science Publishers North-Holland, 1989 pp. 23–73.

[7] Choppy, C., P. Poizat and J.-C. Royer, *A Global Semantics for Views*, in: T. Rus, editor, *International Conference on Algebraic Methodology And Software Technology*, LNCS **1816** (2000), pp. 165–180.

[8] Choppy, C., P. Poizat and J.-C. Royer, *The Korrigan Environment*, Journal of Universal Computer Science **7** (2001), pp. 19–36.

[9] Oheimb, D. v., *Interacting State Machines: a Stateful Approach to Proving Security*, in: A. Abdallah, P. Ryan and S. Schneider, editors, *International Conference on Formal Aspects of Security*, LNCS **2629** (2002), pp. 15–32.