Ramos *et al. Journal of Software Engineering Research and Development* (2015) 3:12
DOI 10.1186/s40411-015-0026-x

Journal of Software Engineering Research and Development
a SpringerOpen Journal

RESEARCH Open Access

Extending statecharts to model system interactions

Marcelo A. Ramos[1], Paulo C. Masiero[1*], Rosangela A.D. Penteado[2] and Rosana T.V. Braga[1]

*Correspondence: masiero@icmc.usp.br
[1]Departamento de Sistemas de Computação - ICMC - Universidade de São Paulo (USP), Av do Trabalhador São-carlense, 400 São Carlos, SP, Brazil
Full list of author information is available at the end of the article

Abstract

Background: Statecharts are diagrams comprised of visual elements that can improve the modeling of reactive system behaviors. They extend conventional state diagrams with the notions of hierarchy, concurrency and communication. However, when statecharts are considered to support the modeling of system interactions, e.g., in Systems of Systems (SoS), they lack the notions of multiplicity (of systems), and interactions and parallelism (among systems).

Methods: To solve these problems, this paper proposes extensions to statecharts. First, a notation to represent a set of orthogonal states, similar in their structures but belonging to different systems, like a pool of telephone systems, is proposed. Second, the concept of communication among parallel states is extended to also represent system interactions, i.e., the relationships among orthogonal systems by means of proper interaction mechanisms like event broadcast.

Results: The proposed extensions to statecharts are symbolic notations that result from an analogy with multi-layer Printed Circuit Boards (PCB). Systems are modeled as concurrent layers that can interact through circuit holes. The resulting diagrams are named pcb-statecharts. Skype-like systems are used to exemplify the modeling of system interactions. They are modeled as concurrent systems disposed in different layers that interact to enable conference calls. A discussion about the use of this notation to model systems of systems is also presented.

Conclusions: The main contribution of this paper is giving to system engineers additional support to model systems interactions. Multiple interacting systems can be designed with separation of concerns. Different viewpoints enable the modeling of these systems as both independent systems and members of a whole. The resulting diagrams improve the notions of multiplicity of systems, and concurrency and parallelism among systems. Additionally, the proposed symbolic notation enables the building of diagrams without the need of physically connecting related entities in the model.

Keywords: Statecharts; Interaction; System of systems; Modeling

1 Introduction

Since the 90's, the complexity of certain solutions has been facing software engineers with the need of designing multiple integrated systems (Maier 1998). One of the challenges of this emerging design paradigm is the appropriate modeling of system compositions, i.e., how systems interact among themselves to achieve a common goal collectively (Brownsword et al. 2006).

Statecharts are diagrams comprised of elements that can improve visually the modeling of reactive system behaviors (Harel 1987). They extend conventional state diagrams with the notions of hierarchy, concurrency and communication. We are particularly motivated by the possibility of using statecharts to support the modeling of system interactions in Systems of Systems (SoS), which are compositions of complex, useful, independent, and interoperable systems that cooperate to achieve objectives that could not be achieved by any of the member systems individually (Boardman and Sauser 2006; Malakuti 2014). Modeling compositions by means of visual elements can increase their understanding and motivate discussions about interactions before they are effectively designed, possibly reducing risks and costs associated to system misbehaviors discovered in advanced stages of the development process.

Statecharts can describe dynamic scenarios where states of a particular system evolve concurrently and coordinately by reacting to stimuli from the system's environment. Differently, in the SoS context, the scenarios comprise interactions among systems that run separately and independently. In this case, each system has its own space of events and its states can evolve by reacting to stimuli from events that come from the whole environment in which they are embedded.

Although several variants have been proposed to statecharts since they were presented (von deer Beeck 1994), most of them aim to overcome problems in the original formalism. Thus, statecharts still lack completeness to model system compositions. Particularly, they lack notions of multiplicity (of systems), and interactions and parallelism (among systems). Indeed, statecharts may not be enough to clearly model SoS scenarios without extensions, e.g., similar states belonging to different systems that interact with each other and evolve independently (Harel and Kahana 1992). These states can be folded by their similarity but will rarely evolve as a unit.

Extensions to statecharts to solve these problems are proposed in this paper: a notation to represent a set of orthogonal states and another to represent the communication among parallel states. The latter extends the concept of communication to also represent system interactions, i.e., the relationships among orthogonal systems by means of proper interaction mechanisms like event broadcast. Moreover, they extend the notions of overlapping to clearly represent a set of orthogonal states, similar in their structures but belonging to different systems, e.g., a pool of telephone systems.

The proposed extensions can be used since the very beginning of the development processes involving multiple integrated systems to build abstract models of these systems and to represent visually their interactions. These models can help engineers, for example, to identify interaction points (Tian et al. 2011), predict communication bottlenecks (Kotov 1997), design collective behaviors, and compose systems (Brownsword et al. 2006). These extensions can be used to represent interactions in Systems of Systems or other types of concurrent and integrated systems.They are exemplified by modeling two versions of a Skype-like system.

The remainder of this paper is organized as follows: Section 2 presents a discussion on why statecharts lack completeness to clearly represent system interactions. Then, in Section 3, symbolic notations are proposed to solve this problem. The results of attaching these notations to statechart elements are discussed as well. Section 4 shows a complete example of how to use the proposed extensions to statecharts to model system

interactions. Section 5 presents related work. Finally, in Section 6, concluding remarks and future work are presented.

2 Modelling system interactions with statecharts

The terms transition and interaction are used distinctly in this paper. Transition is a direct relationship between two states of a system. It is triggered by an event from the system's environment and may change the current active state. On the other hand, interaction is an indirect relationship of different systems. It is an output of a system produced by a transition that is broadcast to other systems which consider it as an event. This event may trigger one or more transitions in these target systems.

Orthogonality describes an *AND* decomposition of states in such way that each part, i.e., the orthogonal states, evolves concurrently (Harel 1987). The notion of communication in statecharts relates to the broadcast of events among orthogonal states of a system (Harel 1987). Figure 1 illustrates an example of event broadcasting (Harel 1988). The initial state is (X, W). An event ω triggers the transition from X to Y and generates an output (which is also an event) λ. Then, λ is broadcast and triggers the transition from W to Z in the orthogonal state. The current state changes to (Y, Z).

Because orthogonal states share a common parent state in the hierarchy, they can share a context. In this case, they can naturally share data and events. On the other hand, states from different systems fit into different contexts, thus, they cannot share data and events without proper mechanisms. Moreover, states from different systems may be concurrent but not orthogonal by definition. Because of these properties, the current statechart broadcast notation may not be applicable to model interactions among systems without extensions. In Fig. 1, for example, if λ could broadcast to other systems, then all the target states should be visually represented as well as the systems they belong to. Otherwise, there is no way to know when λ broadcasts either into or beyond the source system's boundaries and the resulting statecharts would lack clarity. This problem was experienced by Harel since the very first paper on statecharts (Harel 1987).

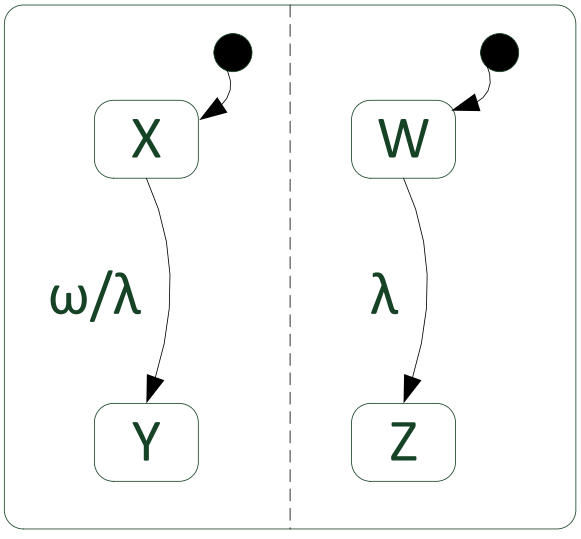

Fig. 1 Event broadcasting (Harel 1988)

Harel (1987) discussed a fairly common scenario in which multiple orthogonal states with the same internal structure can interact to accomplish specific goals. His attempt to model such scenario using statecharts resulted in the 3D diagram shown in Fig. 2. It sketches a phone call by means of interacting states from different but similar parallel states, i.e., telephones. Multiple parallel states with the same structure are represented and each of them can eventually behave as either a caller or a receiver to accomplish a phone call, i.e., the goal. Zave and Jackson have also studied this problem in the context of telecomm systems (Zave and Jackson 1998). In both cases, the problem observed is the same, i.e., it may not be practical to model in detail all the involved communication states without a succinct notation.

Particularly in the context of statecharts, the described problems remain unsolved. Indeed, statecharts still lack proper notations to clearly model multiplicity, parallelism and communication among systems in more complex scenarios like SoS. This is clear in Fig. 2, where artifices like a 3D diagram with arrows crossing planes and ellipsis ("...") were used to represent communication among multiple concurrent states. Certainly this is not a solution since the model can become complex very fast even for simple goals like making audio calls.

The statechart of Fig. 2 fully exemplifies the scenario we are interested in modeling. It highlights the constraints of the current visual elements to support the modeling of interactions among multiple concurrent (sub)systems. An example is the hidden overlapping state "conversing with i" in *telephone-j*. Moreover, "receiver j replaced" is not an event perceived directly by *telephone-i* since the telephones run in different contexts. Instead, it results from interaction mechanisms able to broadcast events beyond the source system's boundaries. Particularly in this case, this event results from *telephone-j* going "on-hook".

In Fig. 3, the use of communications and interactions is exemplified in the context of system interactions. Systems 1 and 2 run independently and some of their states communicate by broadcast mechanisms. Let us suppose that the current state configuration

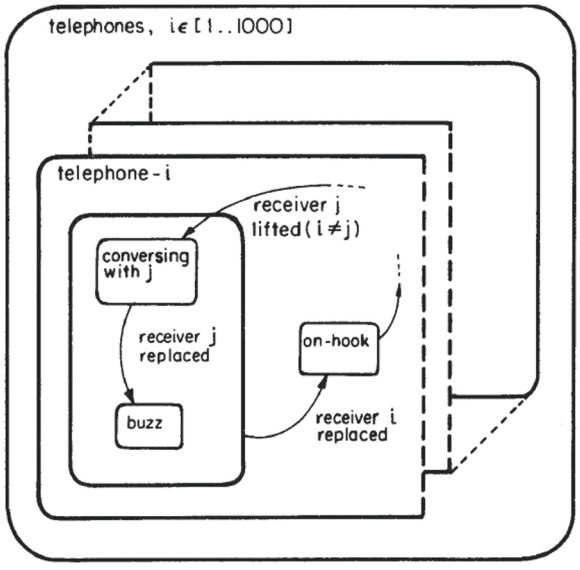

Fig. 2 Parameterized- and states (Harel 1987)

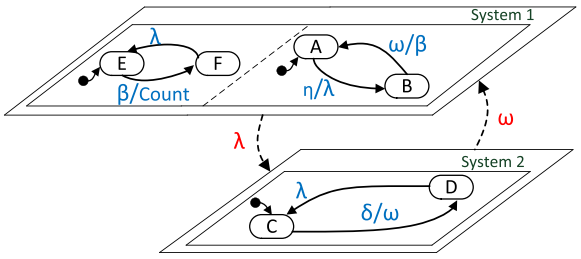

Fig. 3 Communications and interactions in the SoS context

of the composition is $((F, A); D)$ (';' separates groups of states from different systems). In System 1, an event η triggers the transition from A to B and generates an output λ. Then, λ is broadcast both internally to System 1 and to the environment. In System 1, it triggers the transition from F to E. Then, λ reaches System 2 that belongs to the environment and triggers the transition from D to C. The current state configuration changes to $((E, B); C)$. This characterizes a communication among the orthogonal states of System 1 and an interaction between System 1 and System 2, represented by the dashed line in Fig. 3. Similarly, in System 2, an event δ triggers the transition from C to D and generates an output ω. Then, ω reaches System 1, triggers the transition from B to A, and generates a output β. Then, β is broadcast to the orthogonal state E and triggers the transition from E to F. The current state configuration returns to $((F, A); D)$. The spaces of events of System 1 and 2 are $\beta, \eta, \lambda, \omega$ and δ, λ, ω, respectively. Notice that λ and ω are used in the model of both systems but they may react differently in each system. System 1, for example, may receive ω from its own environment and this will not affect the states of System 2.

In the context of system interactions, environmental elements, like events and data, are not shared among systems naturally. In the given example, System 2 is not aware of the counting being performed by System 1. This is because the event β and the counter data are intrinsic elements of System 1. Certainly, these elements can be shared by means of proper interaction mechanisms. The interaction among systems can be considered a cooperative concurrency, because the system that receives an event broadcast from another system will trigger a transition only if its current state is ready to accept the event. Otherwise it will not change.

Mikk et al. (1997) formally describe a statechart as follows: "A finite hierarchy of states, an initial state and a set of transitions. State hierarchy is a tree of states. Nodes of the tree are typed by elements of the set AND, OR, BASIC. Transitions are labeled by a pair consisting of a trigger and an action". This description may not apply directly to the scenario represented in Fig. 3. Indeed, a root state may not exist as a common ancestor for systems 1 and 2 since they run autonomously in separate environments. Thus, λ and ω cannot be described as regular transitions between nodes of a single tree.

Overlapping (Harel and Kahana 1992) does not apply directly to our problem as well. It is inspired in a situation in which different hierarchies of states share a common state, mostly because of the conceptual similarities between the involved states (Harel 1988). Figure 4 shows the overlapping and orthogonality of states. States C and D are overlapping states shared by the parent states $A1$ and $A2$. Additionally, states $A1$ and B are orthogonal states that evolve concurrently. In Fig. 3, systems 1 and 2 may have similar states but they

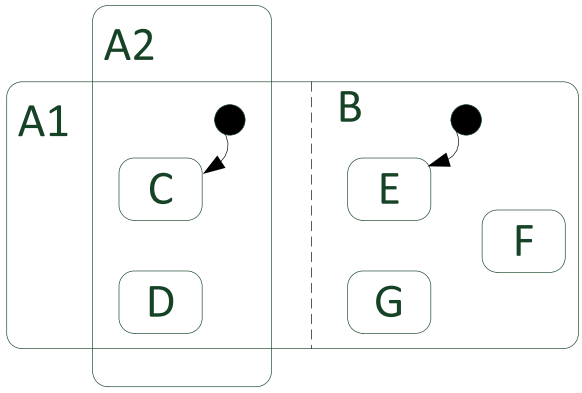

Fig. 4 Overlapping and orthogonality of states (Harel and Kahana 1992)

do not overlap by the given definition. Indeed, states in separate independent systems will rarely share a common state.

Considering what was just discussed about the modeling of system interactions, we show in Fig. 5 a possible way to use the overlapping notation to represent similar states belonging to different systems.

Systems A-1, A-2, and A-3 have similar states, namely *S-1*, *S-2* and *S-3*, respectively. System A-1 has two orthogonal states. One is intrinsic to A-1 and comprises the states $\{D, E\}$. The other, *S-1*, has a structure similar in all other systems and comprises the states $\{A, B, C\}$. Systems A-2 and A-3 are composed similarly. Notice that although we can say that all *S-i* states overlap, they are really distinct instances that can evolve independently in the systems they belong to. The same notice applies to orthogonality. In Fig. 3, for example, states from Systems 1 and 2 can be concurrent but they are not orthogonal. In both cases, extensions to the current notations could improve statecharts and help engineers to clearly model these cases.

The lack of notion of parallelism among systems makes it difficult to represent system compositions in a plain diagram. Additionally, the lack of notion of multiplicity of systems and actions hinders, for example, the representation of how many systems are affected by

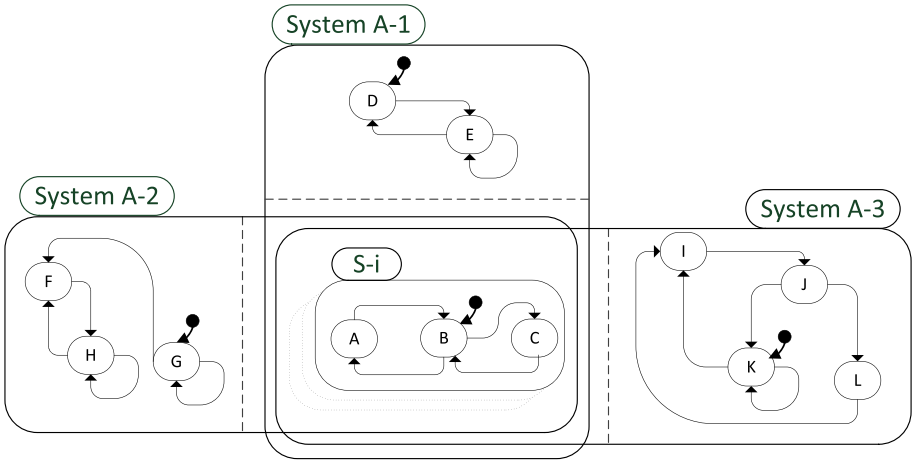

Fig. 5 Overview of several systems with similar states

an event/action and how many systems have a similar state. Also, the lack of notion of interaction makes it difficult to model relationships among states of different systems (see the "..." in Fig. 2).

3 Statechart extensions

Despite the constraints discussed in Section 2, Fig. 2 allowed us to have insights into a solution to model system compositions. In the context of our work, a composition of (sub)systems can be represented by a superposition of distinct planes that interact, like in Fig. 3. The contents of these planes can be similar, e.g., the telephones of Fig. 2, or completely different, e.g., the member systems of a System of Systems (Erl 2008; Lewis et al. 2011; Maier 1998). In both cases, the whole contents of the planes or just part of them may eventually be similar. In this particular case, we can represent all of them using a generic state, as *telephones* in Fig. 2. Then, each particular instance can be represented by parameterized states, like *telephone-i*. *S-i* in Fig. 5 represents the same scenario but the fact that each instance is just part of the corresponding system. Eventually, the number of parameterized states may change dynamically, i.e. change during the system execution. For example, telephones can be added to expand a telephone system or removed for maintenance. Similarly, systems are free to enter and leave a SoS in an uncoordinated way.

Notice that parameterized states conceptually do not overlap. Indeed, they are not shared among different planes and each instance can evolve autonomously in its own environment.

Next, we propose a set of symbolic notations towards a solution to the problems discussed above. They result from an analogy with multi-layer Printed Circuit Boards (PCB) (Khandpur 2005), which are widely used by the electronic industry to connect electronic components by means of multiple circuit layers. In this environment, communication among layers is made via holes. Figure 6 illustrates internally a PCB with 3 layers and two holes. A Through hole goes all the way through the board and connects all layers. Blind via and Buried via connects two or more, but not all, layers of the board.

In our analogy, each layer (plane or system) contains a set of components (states) connected by circuits (transitions). Components from different layers can be connected by holes (interactions). Signals (events) flow through circuits and holes concurrently and coordinately (relationships), thus the components can accomplish a common goal collectively. This scenario describes a composition of systems.

The statechart extensions proposed to model composition of systems and their interactions result when new symbolic notations are attached to statechart elements, specifically

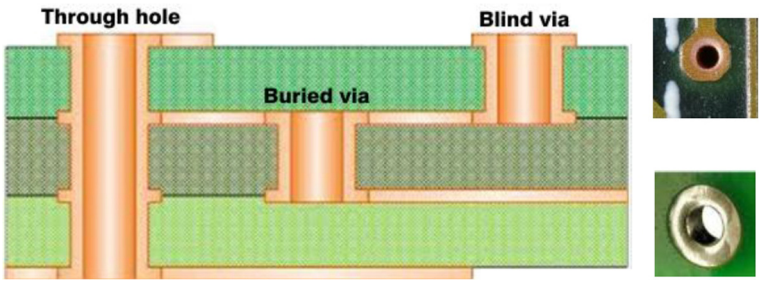

Fig. 6 Multi-layer printed circuit board (PCB)

to states and events. Because of the analogy with PCB, the term **pcb-statechart** is used in this paper to refer to statecharts that contains at least one of the proposed extensions. Generically, each state of a pair of states belonging to different layers is called **layered state**, or simply **l-state**. At the highest level, every layer, i.e., a plane or system, is an l-state when belonging to a composition. Finally, Harel's parameterized states are called **p-states**. In Fig. 5, systems *A*-1, *A*-2 and *A*-3 are l-states and *S-i* are p-states. In Fig. 2, the systems are fully similar, thus, *telephone-i* are both l-state and p-state. As discussed previously, p-states conceptually do not overlap.

It is important to notice that p-states are a particular case of l-states that can be folded because of their similarity. With such definitions, we can define interaction as a relationship among l-states using appropriate mechanisms. In Fig. 3, for example, states $\{A; D\}$ and $\{B; C\}$ are l-states that interact by means of λ and ω, respectively. System 1 and 2 are l-states as well. There are no p-states. It must be clear that relationships among l-states and p-states are not mandatory.

3.1 Statechart extensions applied to systems

The first statecharts' extension results when symbolic notations are attached to p-states. The resulting notations extend statecharts with the notions of multiplicity and parallelism of systems. The notation envisaged by Harel is extended by introducing a 3-tuple $[m, n, F/D]$ with m and n integers and $n \geq m$ specifying the minimum and maximum number of p-states, and an additional information denoting whether the composition, once first established, remains with a fixed number of instances (F) or can change dynamically (D) within the boundary limits $[m..n]$. Some examples are:

- $[4, 4, F]$: composition established with exactly four systems.
- $[3, 10, F]$ composition established with a fixed number of systems between 3 and 10.
- $[2, n, F]$: composition established with a fixed number of at least 2 systems.
- $[m, n, D]$: composition can change dynamically and may comprise a finite but undefined number of systems.
- $[m, 20, D]$: composition can change dynamically and must comprise at most 20 systems.

The pcb-statechart of Fig. 7 represents a hypothetical phone system comprised of exactly four phones. Each instance *Phone-i* is a p-state and also an l-state representing a distinct system.

If the above system is required to be expandable and comprise at least two phones, the resulting pcb-statechart can be the one of Fig. 8.

We tried to preserve the syntax and semantics of regular statecharts regarding entering and leaving p-states. Entry is usually triggered by a transition finishing at the border of a p-state. This means that all instances will be activated as well as their initial states. This is shown by the transition from *F* to *Phone-i* in Fig. 9. The same occurs to enter directly to an internal state as the one from *G* to *B*. In this case all instances will be activated and have *B* as their initial states. Leaving a p-state from the border or from inside, as to states *D* and *E*, is similar to entering and causes all instances to be left. Although syntactically and semantically legal to be modelled, we have found little use for this notation when each instance represents a whole, self-contained system. A possible scenario would be the occurrence of global environmental events that affect equally the whole composition like

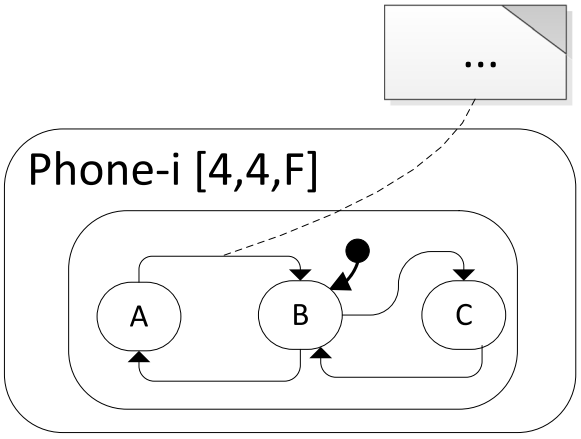

Fig. 7 A pcb-statechart for a phone system with exactly four phones

a blackout that may force all systems to shutdown. When the energy is reestablished, all systems start, for example, by launching recovering procedures.

When the composition of p-states is dynamic, a new situation takes place because instances can enter and leave the composition after it has been first established. A special event *Enter* can harmoniously "introduce" an instance into the composition. Similarly, a special event *Quit* enables an instance to leave harmoniously the composition. These events are illustrated in Fig. 9. The arrow can stop at the border of the state or cross it and reach an internal state; this follows the traditional semantics of statecharts. Although we are exemplifying p-states, it is important to notice that the events *Enter* and *Quit* may also apply to l-states to support compositions of different systems.

When compositions of l-states are generically considered, new problems arise. With the absence of p-states, all l-states are unique, i.e., they cannot be folded. In this case, if they all fit the same model the clarity may decrease fast, mainly when they interact, like in Fig. 3. The proposed solution for this problem also comes from the analogy with PCB. In this case, each layer can be modeled separately, one at a time. When composition is part of the requirements, the associated interactions must be represented. Considering PCBs,

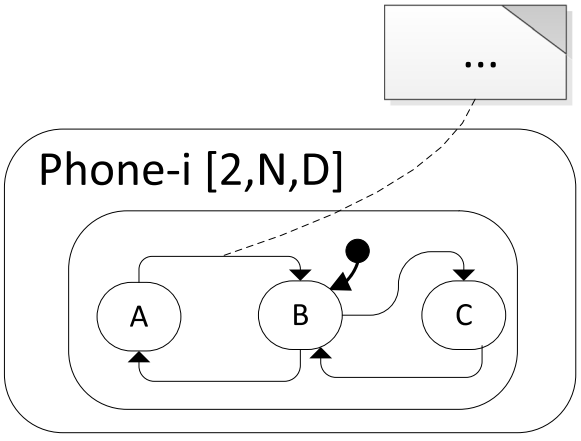

Fig. 8 A pcb-statechart for an expandable phone system

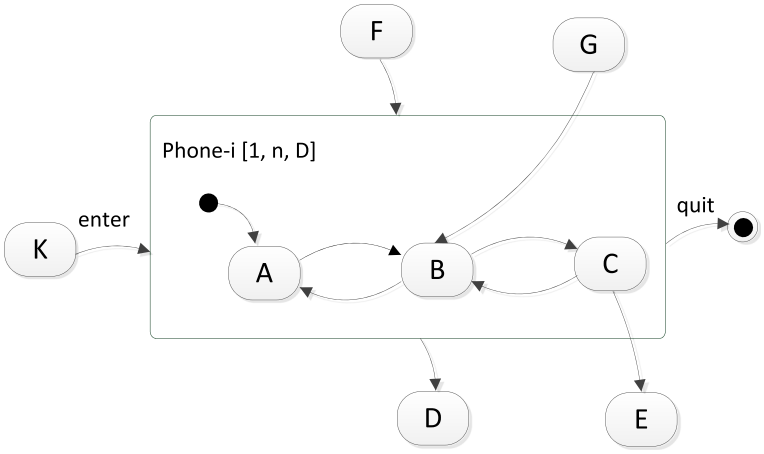

Fig. 9 Entering and leaving p-states

interactions entering and leaving each layer can be easily identified by the holes. Thus, if a similar notation can be applied to pcb-statecharts, it will be possible to represent interactions with the same clarity.

3.2 Statechart extensions applied to events

The major goal of creating symbolic notations to extend events in statecharts is improving the ability to represent system interactions without the inconveniences caused by the use of arrows, discussed previously. Table 1 shows the statechart extensions with the proposed symbolic notations, which are attached to events. The resulting notations represent the holes (interactions) that can be seen when looking to a layer of a composition. Thus, looking to a layer at a time and without using arrows it is possible to observe interactions with a certain level of detail. For example, it is possible to identify interaction points and analyze if and how an event affects other systems. This ability can help engineers to create, model, and validate collective behaviors comprising several systems and interactions. In the SoS context, for example, they could exercise different candidate systems and interactions to achieve the SoS goal before making design decisions.

In Fig. 3, incoming and outgoing signals (events) are placed respectively at the left and right side of the separator /. Thus, the notation ω/λ is clear in representing ω as an incoming event and λ as an outgoing event. This way, it would be enough to attach the proposed symbolic notations to events to improve the notions of multiplicity, interaction, and parallelism. However, an optional short arrow is suggested to emphasize outgoing (arrow

Table 1 Statechart extensions attached to events

Extension	Outgoing	Incoming	Semantics
Broad	e▸○		Send *e* to all other systems via broadcast
Broad		○▸e	Receive *e* from other system via broadcast
Multi	e▸ⓜ		Send *e* to a set of systems via multicast
Multi		ⓜ▸e	Receive *e* from other system via multicast
Uni	e▸①		Send *e* to a single system via unicast
Uni		①▸e	Receive *e* from other system via unicast

entering the hole) and incoming (arrow leaving the hole) events as shown in Table 1. Hereafter, they will always be used in the pcb-statecharts.

Syntactically, symbols from Table 1 are attached to transitions whenever the origin and/or destination of the associated events are other l-states. Because of the terms used in the computer networking area, the proposed extensions presented in Table 1 were named broadcast, multicast, and unicast. They visually represent interactions of a source system with all, many, or one (target) system, respectively, as exemplified in Fig. 10.

The broadcast of an event to orthogonal states can still be represented by simply not attaching an extension to it. Using a short arrow in this case is optional as well.

These three notations cover most of the situations that we would like to specify, but they can leave the model a bit underspecified. Indeed, the notations proposed in Table 1 do not specify to/from which l-states the events are sent/received. Whenever necessary, more precise information about the origin/destination of the events can be described, for example, by using set theory or OCL (OMG 2012). Such descriptions can be conveniently attached to the associated events, for example, using UML notes like in Figs. 7, 8 and 11.

The semantics of the proposed notation is explained informally using an example. Suppose four l-states representing four similar phones belonging to a composition whose folded version is presented in Fig. 7. The same example is represented in Fig. 11 by four unfolded p-states separated by waved dotted lines. We used this notation to make parallelism among l-states distinct from parallelism among orthogonal states (straight dotted lines). In Fig. 11, each p-state belongs to a separate layer, thus, they do not have common ancestors or even constitute an hierarchy. Notice that a UML note is attached to all multicast transitions from *A* to *B* to specify that the event *s*2 is sent only to the "neighbors" of the source system, i.e., the next and the previous systems considering the index assigned to each instance. Notes like this are preferably attached to folded pcb-statecharts but can be attached to unfolded pcb-statecharts as well.

A simulation of the example is presented in Table 2. It shows local events and their effects in a particular instance. Line 2, for example, describes the event *e*1 perceived exclusively by *Phone-3*, the resulting transition from *B* to *C*, and the broadcast of *s*1 to all other instances. We show in a random order that phones 1, 2, and 4 receive *s*1 and react appropriately. Each of them acknowledge the receipt of *s*1 by interacting unicast with the sender, i.e., *Phone-3*. Then, the counter *ct* is incremented in each of these three target instances (lines 3, 4 and 5). Notice that *ct* is particular to each instance.

Next, the occurrence of *e*3 causes *Phone-2* to change from state *B* to *A* (line 6). Then, *e*2 occurs in the context of *Phone-2* (line 7) and *s*2 is sent multicast to its neighbors, i.e., *Phone-1* and *Phone-3*. *Phone-1* receives *s*2 and does nothing because it is in state *B* (line

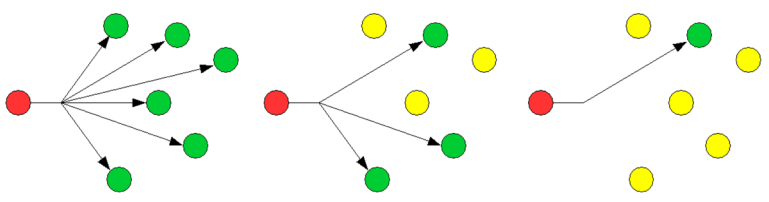

Fig. 10 Broadcast, multicast and unicast interactions

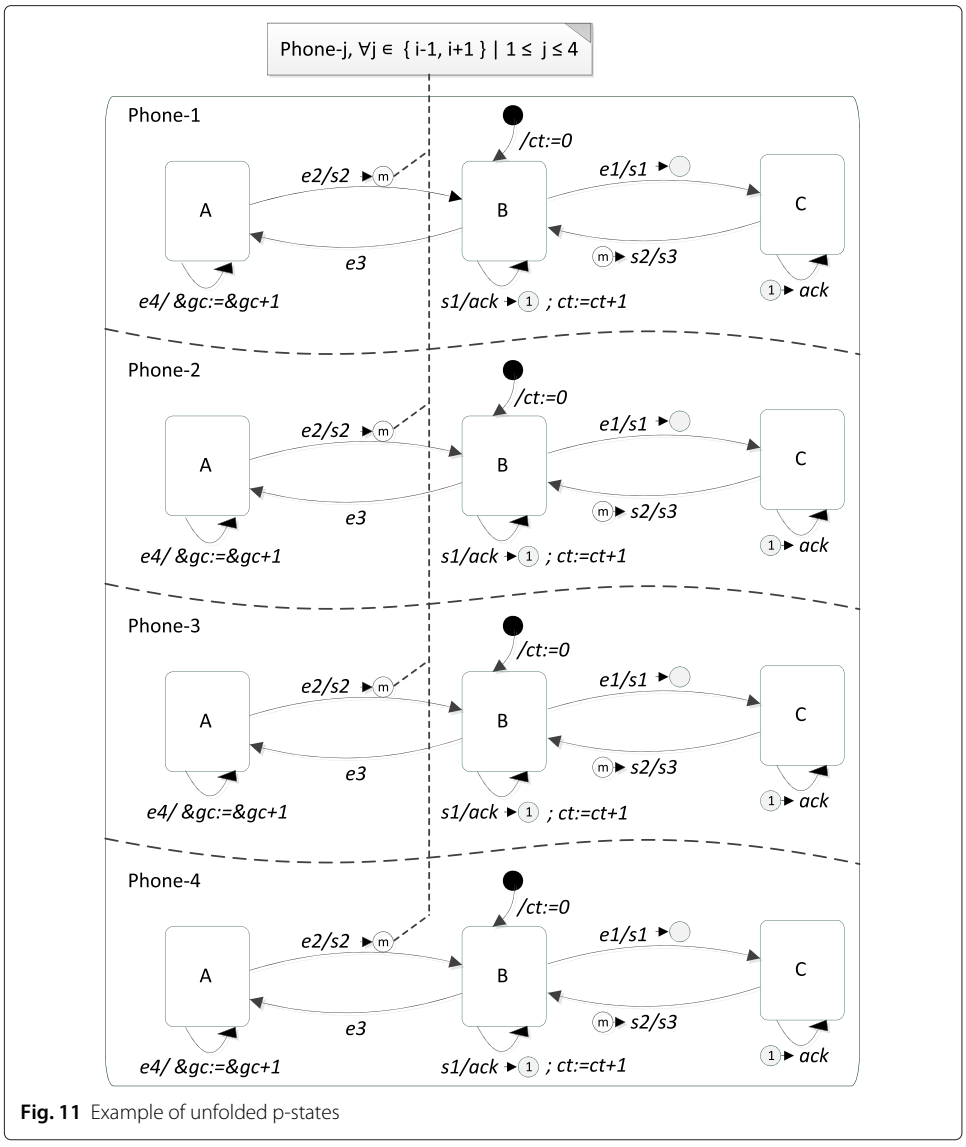

Fig. 11 Example of unfolded p-states

8), but *Phone-3* is in state *C* and changes to *B* triggering *s*3 (line 9). Next, the occurrence of *e*3 causes *Phone-4* to change from state *B* to *A* (line 10). Finally, *Phone-4* receives *e*4 and executes an action to increase the shared counter &*gc*, but it stays in state *A* (line 11). The last step highlights the possibility of sharing data among l-states. The '&' symbol denotes shared data.

It is important to notice that although there are four similar states, they evolve independently. Thus, although they can be folded like in Fig. 8, i.e., *Phone-i*, they do not really overlap according to Harel's definition (Harel and Kahana 1992).

Variations of the symbolic notations proposed in Table 1 can be used to solve ambiguous situations. In Fig. 12, for example, an empty arrow emphasizes that a *PABX* forwards a ring event to a system (the receiver) other than the one that originated the dial (the caller). However, this can be too succinct and lack clarity. Thus, explanatory notes can always be used, as illustrated in Fig. 12.

Table 2 Example - Simulation - 1

#	Events	Configuration	Output	Action
1	init	(B; B; B; B)	-	-
2	Phone-3/e1	(B; B; C; B)	s1	-
3	Phone-1/s1	(B; B; C; B)	ack	Phone-1.ct++
4	Phone-2/s1	(B; B; C; B)	ack	Phone-2.ct++
5	Phone-4/s1	(B; B; C; B)	ack	Phone-4.ct++
6	Phone-2/e3	(B; A; C; B)	-	-
7	Phone-2/e2	(B; B; C; B)	s2	-
8	Phone-1/s2	(B; B; C; B)	-	-
9	Phone-3/s2	(B; B; B; B)	s3	-
10	Phone-4/e3	(B; B; B; A)	-	-
11	Phone-4/e4	(B; B; B; A)	-	&gc ++

3.3 Example of the extensions applied to systems and events

The example of Fig. 13 mix l-states and p-states to model a composition of a single PABX system and several similar phones. The PABX (l-state) runs orthogonally (waved doted line) to the phones (p-states). Numeric identifiers were added to the transitions only to help on pointing to specific elements of interest. They do not belong originally to pcb-statechart diagrams. The symbolic notations of Table 1 are used to represent interactions among layers, instead of arrows like in Fig. 3. It is important to notice that each *Phone-i* instance can be either a caller or a receiver. Therefore, *Phone-i* is fully modeled with both roles represented together. The resulting pcb-statechart comprises states and interactions of interest for the roles of "caller" and "receiver", flattened in a single view. This is a contribution to solve Harel's problem of modeling system interactions, illustrated in Fig. 2.

A successful scenario could be described by a caller as:

1. *Phone-i* system is *Idle* (default).
2. The handset is lifted to start a call.
4. A dial tone is heard.
5. An extension number is dialed.
10. The call is completed.
12,13. The talk takes place.
15. The handset is replaced to end the call.

Additionally, the same scenario could be described by a receiver as:

1. *Phone-i* system is *Idle* (default).
7. A ring is heard.
8. The handset is lifted to answer the call.

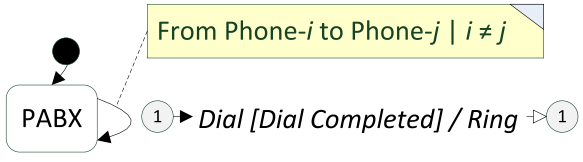

Fig. 12 Solving ambiguous situations with variations

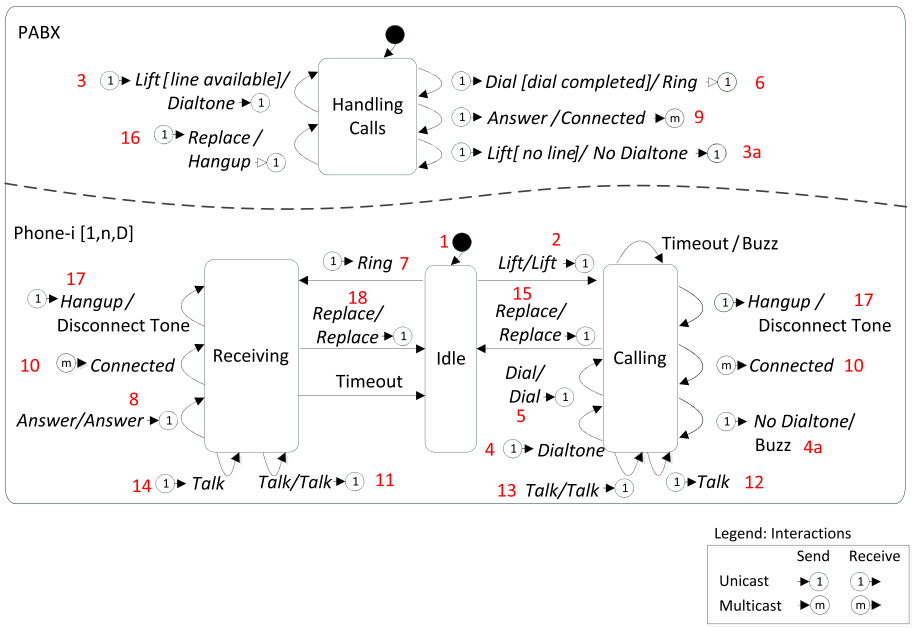

Fig. 13 pcb-statecharts representing a PABX system and several similar phones

10. The call is completed.
11,14. The talk takes place.
17. The call hangs up.
18. The handset is replaced.

Finally, the same scenario could be described by PABX as:

3. A handset is lifted. A dial tone is generated.
6. An extension number is dialed. A Ring is generated.
9. A call is answered. A connection is established.
16. A handset is replaced. The related connection is closed.

Notice that if the above scenarios are merged, the result is a collective behavior that supports a common goal, i.e. enabling two people to talk. This complies with our objective of modeling SoS using pcb-statecharts. Indeed, when the SoS goal is defined, candidate systems can be modeled separately, interactions can be designed to support collective behaviors, and the whole SoS operation can still be described by merging individual behaviors.

Some of the previous steps deserve further explanation.

Step 3: Two filled arrows mean that the incoming and outgoing interactions occur with the same system. In this case, PABX detects the event Lift caused by the lift of the handset in the caller system (Step 2) and, then, returns a dial tone signal to the same system resulting in Step 4.

Step 5: The dial activity is hidden since it is irrelevant to the analysis of the composition. Later, it could be detailed separately from the analysis model by unfolding the Calling state as shown in Fig. 14. This might be a good practice to increase the understanding of the analysis model by keeping the focus on the interactions.

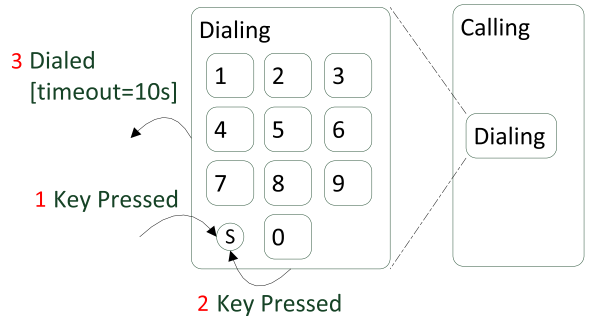

Fig. 14 Unfolding the dial activity

In Fig. 14, a dial starts when a key 0–9 is pressed (1). The triggered transition is directed to the proper state by the selection connector (Harel 1987) according to the key pressed. This process repeats (2) until a delay of 10 seconds since the last key was pressed (3) is detected.

Step 6: An empty arrow means that the incoming and outgoing interactions occur with distinct systems. This case was discussed previously in this section (see Fig. 12) and results in Step 7.

Step 9: The PABX system detects the event Answer caused by the lift of the handset in the receiver system (Step 8) and forwards the event Connected (Step 9) to both the caller and the receiver systems (multicast) resulting in Step 10. This is why Step 10 is represented twice in the diagram. However, it does not mean that a system will receive the event Connected twice. Indeed, the Calling and Receiving states are mutually exclusive in the same layer and just one of them will be handling events in a certain moment.

Many exceptional scenarios, i.e. scenarios with unsuccessful results, can be easily extracted from the diagram of Fig. 13. In the following we present a possible scenario described by the caller:

1. Phone-i system is idle (default)
2. The handset is lifted to start a call.

4a. A buzz is heard (no dial tone)

15. The handset is replaced to end the call

3.4 Semantics and expression power

Pcb-statecharts enable designers to represent more clearly scenarios comprising system interactions. Figure 15, for example, illustrates a pcb-statechart possibly equivalent to the statechart that Harel conjectured in Fig. 2. Notice that the proposed extensions change neither the expressive power nor the semantics of statecharts. Instead, these extensions improve their conciseness power.

The given example shows how the proposed extensions empower the semantics of statecharts. Indeed, Fig. 15 makes clear that every phone can enter and leave the composition dynamically and interact with each other to enable talking. The syntax and semantic of the extensions have been explained using examples. Anyway, a formal specification is desirable and can be explored in a future work.

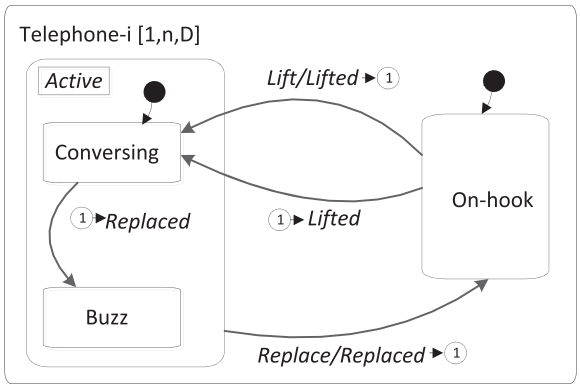

Fig. 15 pcb-statechart similar to the one presented in Fig. 2

4 Modeling system interactions - a more complete and complex example

This section shows other features of the proposed extensions which can be used to model system interactions. The given example comprises Skype-like systems that extend the Phone system of Fig. 13 by allowing, for example, conference calls and chats. This case study illustrates the use of pcb-statecharts in a more general context where several systems running independently and concurrently can interact freely with each other in sets of two to several participants according to known rules and communication requirements. We want to demonstrate that the resulting pcb-statechart can still be clear and understandable while regular models using arrows to connect end points are expected to be confused diagrams as in Fig. 2.

The use case diagram of Fig. 16 shows the functionalities that are discussed in more detail. It is important to note that only a small subset of the Skype-like system functionalities are used to illustrate the extensions proposed. Several events were omitted to keep the example simple.

Different from eliciting requirements of conventional systems that run standalone, react to stimuli from bounded environments, and do not affect other system behaviors, communicating systems like Skype need to be described additionally in terms of interactions.

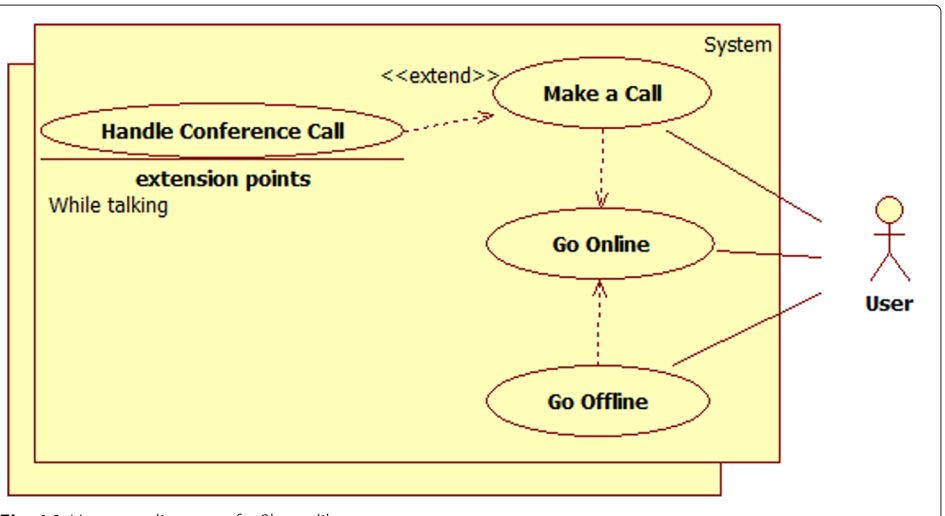

Fig. 16 Use case diagram of a Skype-like system

In fact, each use case of Fig. 16 comprises at least one interaction with other system that affects somehow the behavior of both systems.

Because the user must be online to interact, the use case Go Online must precede all the others. In the diagram, this is represented by the dependency relationships connecting Make a Call and Go Offline to Go Online. The extension point defines that Handle Conference Call is performed whenever a user makes a call to other user, other than the ones that he/she is currently talking to.

When a user goes online, his/her contact list is updated with the most recent Online Status of each contact of the list. Similarly, the contact lists of his/her contacts that are currently available online are updated with the actual user's Online Status. Thus, the post-condition of the Go Online use case would be: "Online Status updated both locally and remotely". It is important to notice that this condition refers clearly to the internal and external effects of the use case.

When a user calls a contact currently available online in his/her contact list, the receiver's system will ring. At this moment the receiver can decide to accept, deny or simply to not answer the call. If the call is accepted, the users can start talking. Then, other users can be similarly added to the call anytime by any participant to handle a conference call. The call ends when all participants have left it either by hanging up or going offline.

When modeling system interactions, engineers usually direct their efforts to the modeling of collective behaviors based on roles and rules, as suggested by Kotov et al. (1997). Roles mostly describe particular capabilities of the participant systems and how they can contribute to achieve more comprehensive goals. Thus, modeling roles often means describing only relevant capabilities of each system and yet in a detail level enough to enable the understanding of their characteristics. On the other hand, rules usually describe interaction mechanisms that every participant system must comply with to support a stable collective behavior. Thus, modeling rules frequently means describing in details when, where, how, and even for how long, the interactions must occur, as well as which systems are involved.

The pcb-statechart of Fig. 17 represents visually the behaviors described previously for interacting instances of a Skype-like system. It includes the proposed extensions, for events and states, and the *Enter* and *Quit* special events (discussed previously in Fig. 9). *Skype-i* denotes that all systems have similar structure (p-states).

All the interactions are of type unicast or multicast because not all Skype-like systems are necessarily represented. Indeed, the number of systems, members of the composition, is limited by the number of contacts in the contact list of each user, which may vary from system to system. That is why it is important to analyse a composition from the point of view of each member. Pcb-statecharts enable such analysis by allowing each system (layer) to be modeled separately, one at a time.

The variables below are used to handle information and create more complex behaviors like in extended state diagrams. These variables have a local scope, i.e., each system has its own set of variables.

#On : Number of contacts available online.
#Tk : Number of users currently talking.
bOff : TRUE means that the initial online status is *Offline*.

Certain functions are invoked during transitions or interactions in Fig. 17. They are:

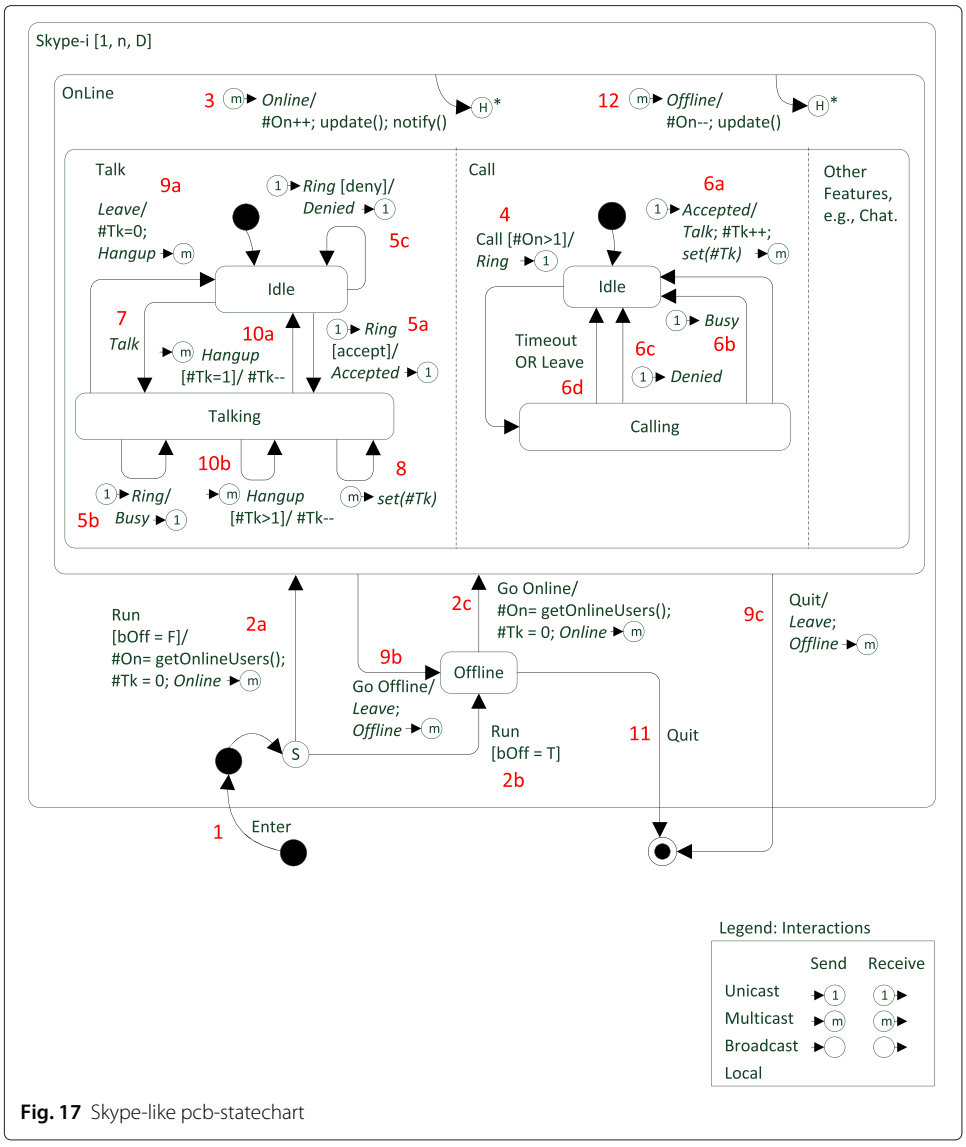

Fig. 17 Skype-like pcb-statechart

update(): Update the local contact list when other systems go online or offline.
notify(): Notify a user when a contact becomes online.
set(): Synchronize the list of participants in a conference call.
getOnlineUsers(): Get the list of users currently online.

When the *Enter* event is triggered, i.e., a system becomes active, a selection connector will direct the system to either the *Online* or *Offline* state, depending on the value of the local variable *bOff*. The *Quit* event will be triggered when a system becomes inactive (9c or 11). In the first case (9c), all member systems are notified, i.e., those systems with a relationship with the system that is quitting the composition. In the second case (11), the system is offline, so the other systems have already been notified before (9b).

Because *Skype-i* represents p-states, we can point to seemingly ambiguous scenarios in the diagram. For example, when a system has this state configuration: *(Talk.Idle, Call.Idle, . . .)*, and the user makes a call from *Call.Idle* (4), one may think that the resulting *Ring* event would be broadcast internally and trigger transitions in orthogonal states, like

Talk.Idle (5a or 5c). That would represent an ambiguous scenario where the user could be calling either himself or another user.

The proposed statechart extensions make such situation unambiguous. The outgoing Unicast extension attached to the Ring event (4) represents that the interaction occurs uniquely and directly with another system that will receive the Ring event exclusively. In this scenario, the notations 5a and 5c relates to events of an orthogonal Talk state in the receiver system. It is important to notice that a Skype-i system can be either a caller or a receiver. This is why the events Call (4) and Ring (5a, 5b and 5c) coexist in the same diagram. On the other hand, when a user (caller) is going to start talking (6a) the Talk event is broadcast to orthogonal states exclusively and will trigger a transition from *Talk.Idle* to *Talk.Talking* (7) in the caller system.

When a participant finishes talking (9a), goes offline (9b), or quits the system (9c) all the remaining participants are notified because the Leave event causes the Hangup event to be sent in multicast. If there are two or more participants still talking ($\#Tk > 1$), the Hangup event just decrements their *#Tk* (10b) and they can keep talking (*Talk.Talking* state active). Otherwise ($\#Tk = 1$), the talk ends (10a) and the last participant returns to the *Talk.Idle* state. In all those scenarios, the participant leaving the talk has his *#Tk* counter set to zero and the *Talk.Idle* state becomes active.

After the pcb-statechart of Fig. 17 is checked thoroughly by the stakeholders and they agree that it is correct, i.e., that it describes visually the expected behavior of a *Skype-i* system, software engineers can go further in describing, modeling, and designing the system. During these processes, the pcb-statechart can be used to support conventional software engineering approaches.

Notice that pcb-statecharts can be used since the very beginning of the development processes. Thus, they are different from those UML diagrams that are used later because they comprise specialized elements such as classes and objects that are not available initially.

Using pcb-statecharts, interactions can be visually represented even when systems are simply abstract rounded rectangles (blobs). This way, they can support different software engineering approaches from requirements to development. Moreover, they can be used, for example, as a reference to textual descriptions like use cases, as exemplified in Fig. 18.

Requirements engineers can embed in their textual descriptions references to the visual elements of a pcb-statechart that represent the same behavior being described. This approach can help, for example, to prevent ambiguities when describing system interactions.

In Fig. 18, the numeric identifiers of the pcb-statechart of Fig. 17 reference specific behaviors in that diagram (see the Main successful scenario). They are embedded in the textual description that focuses mainly the behaviors that comprise at least one interaction among systems. Intrinsic behaviors are omitted intentionally and only successful scenarios are described. The use case Handle Conference Call extends the use case of Fig. 18 whenever the caller Makes a call while he/she is currently talking (Precondition). If succeeded, the receiver is added to a conference call (Postcondition).

The pcb-statechart of Fig. 17 is also full of details to support the development phase. Variable *#Tk*, i.e., the counter for users currently talking, is incremented in (6a) and synchronized in (8). However, it does not mean that *#Tk* is updated twice in a system

UC - Make a Call	
Scope: Skype™-like system **Level**: User goal **Primary actor**: Skype-*i* user (caller) **Stakeholders and Interests**: - Skype-*i* users (Caller and Receiver): . Want to be able to talk to each other **Preconditions**: - Caller and Receiver are online. - Caller is currently not talking. **Postconditions**: Caller and Receiver talk to each other **References**: PCB-statechart of Figure 17	
Main successful scenario	
1	Caller: Calls a contact of his/her contact list. (4)
2	Receiver: Accepts the call. (5a) The caller is notified immediately.
3	Caller: Prepares for talking (6a). Updates the number of people talking
4	Caller: Stars talking (7).
5	Receiver: Updates the number of people talking (8)
6	Caller or Receiver: Leaves the conversation. (9a) The partner system is notified immediately.
7	Caller or Receiver: Hangs up. (10a)

Fig. 18 Use case: make a call

when a call is made. Actually, it is updated just once in each system participating of the call. This is because the related events, i.e., Accepted and Set, occur in different systems. While Accepted occurs in the caller's system (6a), Add occurs in all other systems (Multicast) currently talking (8). Thus, every time a new user enters the call (or conference call) all the participant systems update their own counter *#Tk*. The same happens when a system leaves the call (9a). A Hangup event (Multicast) will cause the decrement of the counter *#Tk* in every remaining system of the call (10b). But, if $\#Tk = 1$, the call ends and the last system returns to the *Talk.Idle* state (10a). A similar synchronism is managed by the variable *#On*, i.e., the counter for contacts currently online. When a user goes online (2a or 2c), offline (9b), or exits the system (9c) all contacts currently online in his/her contact list are notified (Multicast) of the new Online/Offline Status (3 or 12, respectively).

Finally, one may say that the pcb-statechart of Fig. 17 does not reflect accurately the behavior of a Skype-like system. For example, that a server receives events and then distributes them appropriately to the systems. Thus, there are not direct multicast interactions among systems like in (2a, 2c, 9a, 9b, and 9c). If such server really exists, or it should exist to improve the system architecture, it could be added to the diagram just like the PABX system of Fig. 13. Then, source-to-targets multicast interactions ($1 \rightarrow n$) should be changed by a pair source-to-server unicast interaction ($1 \rightarrow 1$) and server-to-targets multicast interaction ($1 \rightarrow n$).

5 Related work

Before discussing related work, it is important to distinguish between studying interoperability mechanisms among systems and modeling interactions. The first intends to identify and classify different forms of interoperability among systems, and to propose solutions to overcome the barriers to achieve them (Morris et al. 2004). Studying interoperability mechanisms is out of the scope of this paper.

In a recent work, Malakuti dealt with problems related to the composition (or integration) of systems to create SoSs (Malakuti 2014). He used as an example the domain of operating systems, with the objective of facilitating adaptivity and energy efficiency. A finite state machine is used to represent each member system. This modeling presents at least two drawbacks: facilitates state explosion and hides composition and integration. These problems are solved at least partially by Statecharts. It is important to notice that Harel has already discussed works related to statecharts in his seminal paper (Harel 1987). This discussion included techniques and languages such as Petri-Nets, ESTEREL, CCS and CSP, as well as the work of Zave (1985) for using finite-state-machines to specify distributed systems. So, we consider that this discussion continues valid for pcb-statecharts as well. Concerning Petri Nets, more recent works looked at using them in a coordinated fashion. Kindler, for example, proposed an Event Coordination Notation (ECNO) based on Petri Net to define a global coordination diagram that coordinates the local behavior of individual parts of the software (Kindler 2012).

Other problem related to what has been discussed in this paper is the dynamic creation and deactivation of parts of processes that are used concurrently to solve a problem. In this direction, Fisher et al. (2011) have proposed a state-transition formalism that supports dynamic reconfiguration and creation/deletion of processes. This is a formal specification and uses reference variables to enable changing the connectivity between processes and referring to instances of processes (Fisher et al. 2011). We have addressed this problem in our proposal by introducing dynamic compositions of systems (see Section 3.1).

Several authors have addressed variations of this problem in the context of UML. Hirsch et al. (2008) proposed the use of a synchronization statechart to model the controller of several collaborating components (Hirsch et al. 2008). They illustrate their approach to model advanced mechatronic systems that form communities of collaborating autonomous agents that can reconfigure dynamically. The communication among components occurs via ports or multi-ports. UML-RT has also been used in this context with components to specify the functional architecture and a complex and hierarchical state-based set of models to specify behavior. van der Beeck et al. (2006) defined a semantic for UML-RT that considers atomic capsules – containing a statechart – and complex capsules that recursively consist of atomic capsules communicating asynchronously with each other over connectors and ports. Co-authors of the same group (Eckardt et al. 2013) have also used the same approach to study how to evaluate properties such as liveness and safety for the whole set of components.

None of these proposals based on statecharts have made modifications or extensions to the concepts and notation of statecharts. The extension proposed in this paper aims at modeling problems based on the concept of several directly communicating processes without a central coordinator. However, it is also possible for a pcb-statechart to be modelled as a coordinator.

Using statecharts to support requirements specification was proposed by Glinz (2002). He explored how a statechart variant for requirements models should look to be as simple as possible, easy to understand and well suited for expressing requirements. Three characteristics were considered essential: 1) typical behavioral and interaction requirements must be expressible with reasonable effort; 2) statechart models, data models and functionality models must smoothly fit together; and 3) state and state transition explosion must be avoided. Truthfully, these statechart variants could extend pcb-statecharts as well. However, there is one important difference. From one system to another, events have to be transmitted explicitly via channels (Leveson et al. 1994) that must provide the information where the event comes from. Figure 19 shows Glinz's integrated object/statechart diagram that provides the requirements for a Room Control of a heating control system.

The symbolic notations proposed to pcb-statecharts can bring several benefits to the understanding of the presented requirements. Mostly, they are more precise than channels to identify interaction points. For example, the point where SetOnOff produces an effect in the room control cannot be affirmed unless intuitively. Thus, if the number of channels grows, the understanding of the model can decrease fast. Just like arrows connecting systems, if all lines are drawn to connect channels to their end points in complex systems, the model may become a puzzle.

6 Concluding remarks and future work

In this paper, we proposed extensions to statecharts to model layered states (l-states), parameterized states (p-states) and their interactions. They result from an analogy with multi-layer PCB and aim to provide the notions of multiplicity (of systems), and interactions and parallelism (among systems). For this purpose, the symbolic notations (Uni), (Multi), and (Broad) were proposed and assigned to statecharts'elements, i.e., states and events. The resulting models were named pcb-statecharts. A Skype-like system has been presented to exemplify the modeling of interacting systems using pcb-statecharts and the benefits were discussed. This proposal to the modeling of system interactions comprises the following characteristics: a) separation of concerns by considering different viewpoints; b) symbolic notation to represent interactions that eliminates the need of connecting related entities in the model; and c) notions of multiplicity of systems, and concurrency and parallelism among systems.

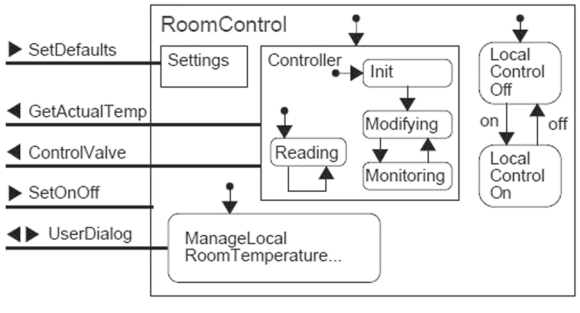

Fig. 19 Requirements for a Room Control (Glinz 2002)

Considering that statecharts are one of the languages of UML, the proposed extensions could be introduced in future versions of UML as well. They could as well be used independently to support the modeling of interactions among systems. Later in the design, when all interactions are completely understood and agreed, details could be included using other modeling resources, available in UML or other notations.

This approach can be particularly useful to model SoS. In fact, SoS engineers are often more concerned about how systems can interact than about how they are built (Brownsword et al. 2006). As the number of SoS members grows and their interactions become more complex it is important to be able to have a big picture of the SoS to discuss, for example, about improvements or the impact of changes in the overall behavior of the SoS. We claim that pcb-statecharts can fit well in this role. Other benefit of using pcb-statecharts to model system interactions is their ability to support both top-down and bottom-up analysis approaches.

Even though we have not performed an empirical evaluation of the proposed approach, some preliminary evidence on the usefulness, benefits and limitations of pcb-statecharts have been gathered from a case study performed during the main author PhD work (Ramos 2014). This case study consisted of a SoS composed of different calculators, publishers and processors systems. The pcb-statechart notation has shown to be appropriate and useful to visually model these requirements and make easier the understanding of the SoS dynamism. We are currently carrying out a controlled experiment to evaluate pcb-statecharts and plan to publish it in the near future.

Despite our particular interest in modeling SoS, we believe that the proposed statechart extensions can also be useful to model other types of systems like those in which interactions among networked systems need to be properly designed before they can communicate. Examples include distributed systems, and service-oriented systems (Lewis et al. 2011; Erl 2008).

Other point to discuss about pcb-statecharts and their benefits relates to their ability to separate concerns. Because they consider one layer per system that interacts it is easy to separate what is relevant or not to each system. Thus, it would be possible to have separated but complementary pcb-statecharts that could be delivered to different teams of system engineers. Later, the diagrams could support those engineers on designing systems able to interact appropriately in the environment modeled by all the pcb-statecharts together.

As for limitations, we can point out that, although the system used as basis for our case study is a real system and was chosen carefully to illustrate the proposed extensions, it may not be comprehensive enough to represent all types of interactions that can occur. However, it was necessary to start with a possibly ideal scenario to propose and validate the new approach.

Ongoing work includes using pcb-statecharts to model SoS of conventional systems like smartphones applications (e.g. WhatsApp, Twitter, retail devices and others). We are also exploring the use of pcb-statecharts to represent the interaction of different software engineering tools. If statecharts and the extensions proposed could be supported by a modeling tool, system interactions could be tested and simulated before any system is designed or definitively chosen. Work on such tool and its usage to support SoS projects is also on the roadmap for future work. Moreover, we want to investigate how easily the

symbolic notations proposed can be understood and applied by students and engineers to describe interactions.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
All authors have participated in the definition of pcb-statecharts and its application to the skype-like example. MAR wrote the first version of the paper, PCM and RTVB wrote the second revised version. All authors read and approved the final manuscript.

Acknowledgements
The authors would like to thank CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) for financial support received during the development of this work.

Author details
[1]Departamento de Sistemas de Computação - ICMC - Universidade de São Paulo (USP), Av do Trabalhador São-carlense, 400 São Carlos, SP, Brazil. [2]Departamento de Computação - DC - Universidade Federal de São Carlos (UFSCar), Rod. Washington Luiz, km 235, São Carlos, SP, Brazil.

Received: 9 October 2014 Accepted: 7 July 2015
Published online: 28 July 2015

References
Boardman JT, Sauser BJ (2006) System of systems - the meaning of *of*. In: Proc. of the 8th Inter. Symposium on service-Oriented System Engineering. IEEE, Los Alamitos, CA, USA. pp 1–6

Brownsword L, Fisher D, Morris EJ, Smith J, Kirwan P (2006) System-of-systems navigator: An approach for managing system-of-systems interoperability. Technical Note TN-019, CMU/SEI:1–39. Available: http://www.sei.cmu.edu/reports/06tn019.pdf

Eckardt T, Heinzemann C, Henkler S, Hirsch M, Priesterjahn C, Schäfer W (2013) Modeling and verifying dynamic communication structures based on graph transformations. Comput Sci- Res Dev 28(1):3–22

Erl T (2008) Sevice-oriented Architecture: Concepts, Techonology, and Design. Prentice Hall, Upper Saddle River, New Jersey, USA

Fisher J, Henzinger TA, Nickovic D, Piterman N, Singh AV, Vardi MY (2011) Dynamic reactive modules. In: CONCUR'11 - Concurrency Theory Conference. Springer Verlag, Berlin Heidelberg. pp 404–418

Glinz M (2002) Statecharts for requirements specification - as simple as possible, as rich as needed. In: Proc. ICSE 2002 Int. Workshop on Scenarios and State Machines: Models, Algorithms and Tools. ACM, New York, NY, USA. pp 1–5

Harel D (1987) Statecharts: A visual formalism for complex systems. J Sci Comput Program 8(3):231–74

Harel D (1988) On visual formalism. Commun ACM 31(5):514–30

Harel D, Kahana C (1992) On statecharts with overlapping. ACM Trans Softw Eng Methodol 1(4):399–421

Hirsch M, Henkler S, Giese H (2008) Modeling collaborations with dynamic structural adaptation in mechatronic UML. In: Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems. SEAMS '08. ACM, New York, NY, USA. pp 33–40

Khandpur R (2005) Printed Circuit Boards: Design, Fabricating, and Assembly. McGraw-Hill, New York, NY, USA. 704p

Kindler E (2012) Modelling local and global behaviour: Petri nets and event coordination. In: Jensen K, van der Aalst W, Ajmone Marsan M, Franceschinis G, Kleijn J, Kristensen L (eds). Transactions on Petri Nets and Other Models of Concurrency VI. Lecture Notes in Computer Science. Springer, Berlin Vol. 7400. pp 71–93

Kotov V (1997) System of systems as communicating structures. Technical report, HP Labs HPL-97-124:1–15. Available: http://www.hpl.hp.com/techreports/97/HPL-97-124.pdf

Lewis G, Morris E, Simanta S, Smith D (2011) Service orientation and systems of systems. IEEE Soft 28(1):158–63

Leveson NG, Heimdahl MPE, Hildreth H, Reese JD (1994) Requirements specification for process-control systems. IEEE Trans. on Software Engineering 20(9):684–707

Maier MW (1998) Architecting principles for systems-of-systems. J Int Council Syst Eng 1(4):267–84

Malakuti S (2014) Detecting emergent interference in integration of multiple self-adaptive systems. In: Proceedings of the 2014 European Conference on Software Architecture Workshops. ECSAW '14. pp 24–1247. http://doi.acm.org/10.1145/2642803.2642826

Mikk E, Lakhnech Y, Petersohn C, Siegel M (1997) On formal semantics of statecharts as supported by STATEMATE. In: Second BCS-FACS Northern Formal Methods Workshop. Springer-Verlag, Berlin. pp 1–14

Morris EJ, Levine L, Place PR, Plakos D, Meyers BC (2004) System of systems interoperability. Technical Report TR-004, CMU/SEI:1–67. Available: http://resources.sei.cmu.edu/asset_files/TechnicalReport/2004_005_001_14375.pdf

OMG (2012) OMG Object Constraint Language (OCL), v2.3.1. Online. Available: http://www.omg.org/spec/OCL/ - last access in 10/06/2014

Ramos MA (2014) Bridging software engineering gaps towards system of systems development. PhD thesis, ICMC-University of Sao Paulo, Brazil. Advisor: Profa. Dra. Rosana T. V. Braga, 134 pgs. Available at: http://www.teses.usp.br/teses/disponiveis/55/55134/tde-13082014-103931

Tian JZ, Wang JZ, Ding HQ, Liang W (2011) Visualizing and Modeling Interaction Relationships Among Entities. IBM Corp. Patent No. US 7930678 B2. USA. http://www.google.es/patents/US7930678

von deer Beeck M (1994) A comparison of statecharts variants. LNCS: Formal Techniques in Real Time and Fault Tolerant Systems 863(1):128–148. Springer-Verlag, New York
von der Beeck M (2006) A formal semantics of UML-RT. In: Nierstrasz O, Whittle J, Harel D, Reggio G (eds). MoDELS. Lecture Notes in Computer Science. Springer, Berlin Vol. 4199. pp 768–782
Zave P (1985) A distributed alternative to finite-state-machine specifications. ACM Trans Program Lang Syst 7(1):10–36
Zave P, Jackson M (1998) A component-based approach to telecommunication software. IEEE Soft 15(5):70–78