A Common Framework for Synchronization in Requirements Modelling Languages

Shahram Esmaeilsabzali and Nancy A. Day and Joanne M. Atlee

Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
{sesmaeil,nday,jmatlee}@cs.uwaterloo.ca

Abstract. The ability to describe synchronization between the components of a model is a fundamental primitive in modelling languages. After studying existing modelling languages, we discovered that many synchronization mechanisms can be organized into a common abstract framework. Our framework is based on a notion of synchronization between transitions of complementary roles. It is parameterized by the number of interactions a transition can take part in, i.e., one vs. many, and the arity of the interaction mechanisms, i.e., exclusive vs. shared, which are considered for the complementary roles to result in 16 synchronization types. We describe how many modelling constructs, such as multi-source, multi-destination transitions, many composition operators, and many workflow patterns are forms of synchronization. By generalizing and classifying synchronization types independently of a particular language, our goal is to enable language designers to adopt an appropriate synchronization type for a domain effectively.

1 Introduction

The ability to describe synchronization between the components of a model is a fundamental primitive in modelling languages. Process algebras, such as CSP [9] and CCS [14], make synchronization the focus of the model. Requirements modelling languages, such as statecharts [8, 18], its variants [2], and UML StateMachines [16], have generally been more centred around behaviour described using events, guards, and actions on variables. The concurrency primitives included in these languages usually have little or no explicit synchronization mechanisms. In some cases, syntax such as multi-source and multi-destination transitions hides a form of synchronization. However, models of systems can often be more precisely and concisely expressed when using explicit synchronization mechanisms.

Inspired by the clean and useful synchronization mechanisms of process algebras, our goal is to understand the role of synchronization in big-step modelling languages (BSMLs). In previous work, we introduced the term BSMLs to describe a popular class of requirements modelling languages, where the model's reaction to an environmental input is described as a big step consisting of a sequence of small steps each with possibly multiple, concurrent transitions [4–6]. We found that while the BSML family has many variations of ways in which generated events trigger transitions in subsequent small steps, in general, they are lacking in the means to describe how transitions should be

synchronized together *within* a small step. In this paper, we extend the family of BSMLs to support different types of synchronization.

In studying the potential use of synchronization mechanisms in BSMLs, we find that there are many different possible synchronization types and that we can organize the design space of these options using a language-independent, parameterized framework. Our framework is based on a notion of synchronization between transitions of complementary roles, analgeous to CCS [14]. Our framework is parameterized by the number of interactions a transition can take part in, i.e., one vs. many, and the arity of the interaction mechanisms, i.e., exclusive vs. shared, which are considered for both complementary roles to result in 16 types of synchronization. In this paper, we focus on the applications of these synchronization types in requirements modelling languages. Our framework allows us to show how many modelling constructs, such as multi-source, multi-destination transitions [8, 18], many composition operators [15], many workflow patterns [1], and a notion of signal [17] are forms of synchronization. Furthermore, our framework gives us a vocabulary for including a variety of types of synchronization in one model. Compared to previous work on classifying synchronization mechanisms, our work describes a different (cf., [10]) or a larger (cf., [15]) set of synchronization types; it allows multiple types to be used within one model; and it is focused on BSMLs.

Our contribution in this paper is threefold. First, we present a framework for classifying synchronization types in a language-independent way. Second, we introduce the family of *synchronizing big-step modelling languages* (SBSMLs) that combine existing BSMLs with synchronization capabilities. Third, we demonstrate the usefulness and generality of our framework by describing how many modelling constructs are forms of synchronization that can be expressed using the synchronization types of SBSMLs.

By generalizing and classifying synchronization types independently of a particular language, our framework allows language designers to consider the synchronization types appropriate for a domain in relation to other modeling concepts found in the domain without historical syntactic and semantic dependencies found between language constructs. Thus, a language designer can choose and adopt an appropriate synchronization type for a language in a domain effectively.

The remainder of the paper is organized as follows. Section 2 briefly describes the syntax and the semantics of BSMLs. Section 3 presents our synchronization syntax in SBSMLs, our framework of 16 synchronization types, and the role of these synchronization types in the semantics of SBSMLs. Section 4 presents how synchronization can describe the semantics of various modelling constructs. Section 5 considers related work, including the comparison of our work with languages that are specialized for specifying deterministic behaviour. Section 6 concludes the paper, laying out the direction of our future work.

2 Background: Big-Step Modelling Languages (BSMLs)

In this section, we present an overview of the common syntax and semantics of BSMLs [4–6]. We adopt a few syntactic definitions from Pnueli and Shalev's work [18].

A BSML model is a graphical, hierarchical, extended finite state machine, consisting of: (i) a *hierarchy tree* of *control states*, and (ii) a set of *transitions* between them.

A control state has a *type*, which is one of *And*, *Or*, or *Basic*. All *Basic* control states of a model, and only they, appear at the leaves of the hierarchy tree of the model. A control state is *compound* if its type is either *And* or *Or*. Each compound control state has a set of *child* control states. A control state that is a child of another control state through transitivity is its *descendant*. Similarly, the *parent* and *ancestor* relations are defined with their usual meanings. In the graphical representation, the children of an *And* control state are separated by dashed lines. Fig. 1 shows an SBSML model that we use to describe the syntax and semantics of BSMLs and SBSMLs. The model characterizes a set of simple synchronized ice skating programs. Initially, all skaters are together, represented by the *Basic* control state *Together*. During the program, the skaters can split into three groups to perform the *intersection* maneuver, represented by the *And* control state *Intersection*.[1] To avoid a clash, at each point of time, only one of the three groups can initiate an intersection maneuver. Control states $Group_1$, $Group_2$, and $Group_3$ are *Or* control states, and are the children of control state *Intersection*. The skaters can merge back into a group, but the program can only end by a transition to the *End* control state, when the skaters are split. Each *Or* control state has a *default* control state, which is one of its children that is graphically signified by an arrow without a source control state; e.g., the default control state of control state $Group_1$ is G_{11}. The *least common ancestor* of a set of control states is the lowest (closest to the leaves) control state in the hierarchy tree such that each of the control states is its descendant. Two control states are *orthogonal* if neither is an ancestor of the other and their least common ancestor is an *And* control state; e.g., $Group_1$ and $Group_2$ are orthogonal.

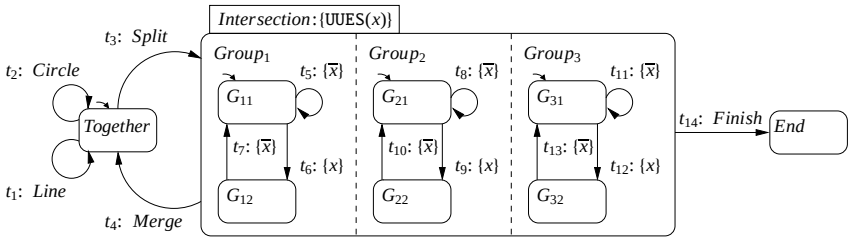


Fig. 1. A model for a set of synchronized ice skating programs.

Each transition has a *source* control state and a *destination* control state. Additionally, a transition can have: (i) enabling conditions, in the form of an *event trigger*, which is the conjunction of a set of events and negations of events, and a *guard condition*, which is a boolean expression over variables; and (ii) a set of actions, in the form of *variable assignments* and *generated events*. For example, in the model in Fig. 1, the source and destination control states of t_3 are *Together* and *Intersection*, respectively; its event trigger is *Split*. A transition is a *self transition* if its source and destination control states are the same; e.g., transitions t_1 and t_2 are self transitions, which represent the *circle* and *line* maneuvers, respectively.[2] Two transitions are *orthogonal* if their source control states are orthogonal, as well as, their destination control states; e.g., t_5 and t_9.

[1] In the intersection maneuver, the skaters in one group skate between the skaters of another.
[2] In these maneuvers, the skaters create a formation in a circle and a line pattern, respectively.

A BSML semantics specifies how the reaction of a model to an *environmental input* is computed as a *big step*. A big step is an alternating sequence of *snapshots* and *small steps*, which ends when there are no more enabled transitions. At a snapshot of a model, there could exist multiple *potential small steps*, each of which can be taken as the next small step. Each small step is the execution of a maximal set of transitions that are enabled and pairwise orthogonal. Each snapshot captures the information about the state of the model, including the control states that the model resides in, the statuses of events, etc., which determine the *enabledness* of a transition. If a model resides in an *And* control state, it resides in all of its children. If a model resides in an *Or* control state, it resides in one of its children. The execution of a small step of a model updates the current snapshot of the model to a new snapshot that captures the effects of executing the transitions in the small step. If the destination control state of a transition in a small step is a compound control state, the default arrows are followed to determine the control states of the next snapshot. As an example, when the model in Fig. 1 resides in control state *Together* and environmental input events *Circle* and *Split* are received, either of the two potential small steps $\{t_2\}$ and $\{t_3\}$ can be taken, arriving at control state *Together* or *Intersection*, respectively. There are many variations in the semantics of how a BSML determines the enabledness of a transition and its execution effects [4–6].

3 Synchronizing Big-Step Modelling Languages (SBSMLs)

In this section, first, we introduce our synchronization syntax for SBSMLs. We then present our parametric classification of 16 synchronization types together with a description of their roles in the semantics of SBSMLs.

3.1 Synchronization Syntax

A compound control state of an SBSML (both *And* and *Or* control states) can have a set of *synchronizers*, which are graphically positioned at the top of the control state. For example, the control state *Intersection* in the model in Fig. 1 has one synchronizer: UUES(x). Each synchronizer $Y(L)$ has: (i) a *synchronization type*, Y; and (ii) a *label set*, L, surrounded by parentheses, instead of curly brackets. There are 16 synchronization types, each of which is a string of four letters, where a letter represents an aspect of the semantics of the synchronization type. The label set of a synchronizer *declares* a unique set of *identifiers* (labels) that are *used* by transitions that are to be synchronized by the synchronizer. In the model in Fig. 1, synchronizer UUES(x) has synchronization type UUES, and declares the identifier x in its label set $\{x\}$.

A transition in an SBSML model can have: (i) a set of *role sets*, and (ii) a set of *co-role sets*. Each role set is a set of *labels*, each of which is an identifier. Each co-role set is a set of *co-labels*, each of which is an over-lined identifier. For example, in the model in Fig. 1, the set of role sets of t_6 is $\{\{x\}\}$ and the set of co-role set of t_8 is $\{\{\overline{x}\}\}$. The well-formedness criteria of SBSMLs, summarized at the end of this section, require that all of the labels (co-labels) of a role set (co-role set) are associated with the identifiers of the same synchronizer. When the set of role sets or the set of co-role sets of a transition is a singleton, its curly brackets are dropped. A role set is called *uni-role*

if it is a singleton and *poly-role* otherwise. Similarly, a co-role set is called *uni-co–role* or *poly-co–role*. For example, the only role set of t_6 is a uni-role. Transitions t_6, t_8, and t_{11} can execute together because synchronizer UUES(x) match their role and co-role sets.

3.2 Synchronization Types and Semantics

A synchronization type consists of a sequence of four letters, each of which is a value for one of the four parameters that together create the set of 16 synchronization types. Table 1 describes the role of each parameter and its corresponding two possible values, when considered for an arbitrary synchronizer $Y(L)$. The "Index" column relates the position of a letter in the synchronization type with its corresponding parameter. Next, we describe the semantics of synchronization types in detail.

Table 1. Synchronization types and their parameters, when considered for synchronizer $Y(L)$.

Index	Parameter Purpose	Values for Synchronizer $Y(L)$
1	How an identifier can be used in the role sets of transitions	U: The identifiers in L can be used only in uni-roles
		P: The identifiers in L can be used in poly-roles
2	How an identifier can be used in the co-role sets of transitions	U: The identifiers in L can be used only in uni-co-roles
		P: The identifiers in L can be used in poly-co-roles
3	How many instances of a label can appear in the role sets of transitions in a small step	E: One, exclusively
		S: Many, in a shared manner
4	How many instances of a co-label can appear in the co-role sets of transitions in a small step	E: One, exclusively
		S: Many, in a shared manner

From a set of enabled, orthogonal transitions, T, determined by the semantics of BSMLs, a potential small step, X, $X \subseteq T$, must satisfy the constraints of all of the synchronizers that control transitions in T.

In a synchronizer $Y(L)$, the first two letters of its synchronization type, Y, indicate how the identifiers in L can be used in transitions within the scope of $Y(L)$. A U in the first position means that for all identifiers $l \in L$, all transitions in X that have l in their role sets, l must belong to a uni-role (i.e., a singleton role set). A U in the second position means that for all identifiers $l \in L$, all transitions in X that have $\overline{l}$ in their co-role sets, $\overline{l}$ must belong to a uni-co-role set. A P in the first or second position of the synchronization type places no such constraints but only has a different meaning from a U if there are multiple identifers in L. The constraints of the first two indices in the synchronization type can be checked syntactically by well-formedness constraints.

As in some process algebras, such as CCS [14], a label in a role set, e.g., m, is *matched* with a co-label in a co-role set that has the same identifier, i.e., $\overline{m}$. For every transition, t, included in X, the labels in all its role sets and the co-labels in all its

co-role sets *must* participate in a match: For every label, m, in a role set, there must be a matching co-label, $\overline{m}$, from another transition included in X, and vice-versa for every co-label, $\overline{n}$, in its co-role sets. The third and fourth indices of the synchronization type indicate how many transitions can participate in this match: Effectively, how many labels, m, can match an $\overline{m}$ and vice-versa, amongst the role sets and co-role sets of the transitions in X. For a synchronizer with label set L and a synchronization type whose third letter is `E`, i.e., one of the `**E*` synchronization types, every identifier, $l \in L$, can appear at most once in the role sets of all transitions in X. For synchronization types `***E`, every over-lined identifier of L, $\overline{l}$, can appear at most once in the co-role sets of all transitions in X. For synchronization types `**S*` (and `***S`), an identifier $l \in L$ can appear multiple times in the role sets (and co-role sets) of the transitions in X.

In summary, after collecting the role sets and co-role sets of all the transitions within X that use identifiers of L, we have a set of role sets and a set of co-role sets:

$$\{\{r_1^1, r_1^2, \cdots\}, \{r_2^1, r_2^2, \cdots\}, \cdots\} \text{ and}$$
$$\{\{c_1^1, c_1^2, \cdots\}, \{c_2^1, c_2^2, \cdots\}, \cdots\}.$$

These sets should satisfy all of the following conditions:

- Every label r_u^v must have a corresponding co-label $c_x^y = \overline{r_u^v}$, and vice versa for every co-label; and
- If the synchronization type is `**E*`, for every co-label c_x^y, there is exactly one corresponding label r_u^v, such that $= c_x^y = \overline{r_u^v}$;
- If the synchronization type is `***E`, for every label r_u^v there is exactly one corresponding co-label c_x^y, such that $\overline{r_u^v} = c_x^y$; and
- Finally, the set X must be maximal, i.e., it is not possible to add one or more transition in T and to satisfy the above constraints of the synchronization type.

Table 2 shows examples of synchronizing transitions according to 10 synchronizers of distinct types. The transitions in each row are enabled, orthogonal transitions. Intuitively, the first two letters of a synchronization type specify the number of interactions, i.e., the number of matchings over distinct identifiers, that a transition can take part in, i.e., *biparty* vs. *multiparty* interaction. The last two letters of a synchronization type specify the arity of the interaction mechanism, i.e., *exclusive* vs. *shared* interaction.

In the model in Fig. 1, when the model resides in G_{11},G_{21}, and G_{31}, the set of transitions $\{t_5, t_9, t_{11}\}$ is a potential small step of the model, which satisfies the constraints of synchronizer `UUES`(x): (1) only uni-roles use x; (2) only uni-co-roles use x; (3) only t_9 has a role set including x; and (4) both t_5 and t_{11} have co-role sets including $\overline{x}$. The other two potential small steps are: $\{t_6, t_8, t_{11}\}$ and $\{t_5, t_8, t_{12}\}$. The model neither permits two groups to initiate an intersection maneuver simultaneously, nor a group to initiate two intersection maneuvers consecutively.

Each pair of synchronization types `UPEE` and `PUEE`, `UUSE` and `UUES`, `UPSE` and `PUES`, `PUSE` and `UPES`, `PPSE` and `PPES`, and `UPSS` and `PUSS` are symmetric. A synchronizer with one of these types can be replaced with a synchronizer with the same label set but the symmetric type, with the role sets and co-role sets of transitions being swapped.

If a model has more than one synchronizer, the constraints of their corresponding synchronization types should be considered together; i.e., the set X above should satisfy the synchronization requirements of all of the synchronizers together.

Table 2. Examples of synchronizing transitions.

Synchronizer	Synchronizing Transitions
UUEE(m)	t_1: $\{m\}$, t_2: $\{\overline{m}\}$
UUSE(m)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{\overline{m}\}$
UUSS(m)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{\overline{m}\}$, t_4: $\{\overline{m}\}$
UPEE(m, n)	t_1: $\{m\}$, t_2: $\{n\}$, t_3: $\{\overline{m}, \overline{n}\}$
UPSE(m, n)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{n\}$, t_4: $\{n\}$, t_5: $\{\overline{m}, \overline{n}\}$
UPES(m, n)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{\overline{m}, \overline{n}\}$, t_4: $\{\overline{m}, \overline{n}\}$
UPSS(m, n)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{n\}$, t_4: $\{n\}$, t_5: $\{\overline{m}, \overline{n}\}$, t_6: $\{\overline{m}, \overline{n}\}$
PPEE(m, n, p, q)	t_1: $\{m, n\}$, t_2: $\{p, q\}$, t_3: $\{\overline{m}, \overline{p}\}$, t_4: $\{\overline{n}, \overline{q}\}$
PPSE(m, n, p, q)	t_1: $\{m, n, p, q\}$, t_2: $\{m, n\}$, t_3: $\{p, q\}$, t_4: $\{\overline{m}, \overline{p}\}$, t_5: $\{\overline{n}, \overline{q}\}$
PPSS(m, n, p, q)	t_1: $\{m, n, p, q\}$, t_2: $\{m, n\}$, t_3: $\{p, q\}$, t_4: $\{\overline{m}, \overline{n}, \overline{p}, \overline{q}\}$, t_5: $\{\overline{m}, \overline{p}\}$, t_6: $\{\overline{n}, \overline{q}\}$

Lastly, in the semantics described above, a few well-formedness conditions were assumed. An SBSML model is *well-formed* if all of the following five conditions hold,

i Each label uniquely belongs to the label set of exactly one synchronizer.
ii No two synchronizers of a control state have the same synchronization type.
iii For each label l of synchronizer m and each transition t, l is associated with at most one of the role sets or co-role sets of t. Furthermore, m is associated with the lowest control state that is an ancestor of the source and destination control states of the transitions that use the labels of m in their role sets or co-role sets.
iv Two labels that are associated with the same synchronization type do not belong to two different role sets or two different co-role sets of the same transition.
v For each label l, if there is at least one transition with a poly-role that includes l, then the synchronization type of its corresponding synchronizer should be a synchronization type whose first letter is P (i.e., P***), otherwise it must be one of the U*** synchronization types. Similarly, the second letter of a synchronization type is specified based on the characteristics of the co-role sets of transitions.

Hereafter, by an SBSML model, we mean a well-formed SBSML model.

4 Applications: Semantics of Modelling Constructs

Through examples, this section describes how the essence of the semantics of modelling constructs such as as *multi-source, multi-destination transitions* [8,18], *composition operators* [15], and *workflow patterns* [1] can be captured succinctly using synchronization in SBSMLs. We also show how some of the semantic variations of BSMLs can be modelled using synchronizers, thereby allowing multiple BSML semantics to exist in different components of a model. Lastly, we show how a notion of a *signal* and the negation of a signal used in some BSMLs can be modelled in SBSMLs.

4.1 Modelling Multi-source, Multi-destination Transitions

Multi-source, multi-destination transitions embody a form of concurrent, synchronized execution: When a multi-source, multi-destination transition is executed, it exits all of its source control states and enters all of its destination control states [8, 18]. A multi-source, multi-destination transition of a model can be replaced with a set of regular transitions that are always taken together by synchronizing via a synchronizer of type UPEE. As an example, the SBSML model in Fig. 2(b) is equivalent to the model in Fig. 2(a), which has two multi-source, multi-destination transitions x and y. For example, transition x is replaced by transitions x_1, x_2, and x_3. One of the transitions, x_1, adopts the enabledness conditions, the actions, and the role sets and co-role sets of x, along with a new co-role set with co-labels for every other transitions. The other transitions each has one singleton role set, to match the co-role set of the first transition. The number of control states in the source and destination of a multi-source, multi-destination transition need not be the same, in which case new dummy control states are introduced to make the number of source control states and destination control states equal. For example, in the model in Fig. 2(b), control state R_4 is such a dummy control state.

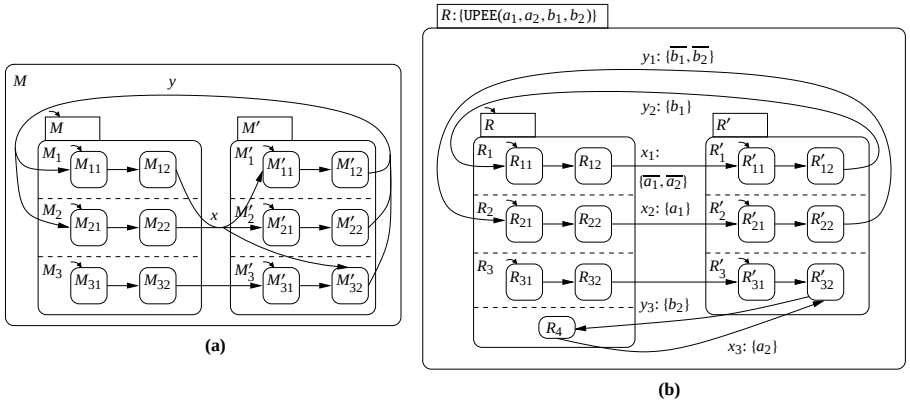


Fig. 2. Multi-source, multi-destination transitions using regular transitions.

4.2 Modelling BSML Semantic Variations

In previous work, we deconstructed the semantics of BSMLs into eight high-level, mainly orthogonal, *semantic aspects* and their *semantic options* [4–6]. These aspects were divided into: (i) dynamic aspects, which determine the enabledness of a single transition and the effect of its execution; and (ii) structural aspects, which specify the possible small steps from a set of enabled transitions [4]. Using synchronization, we show that the semantic options for the structural semantic aspects of *concurrency* and *preemption* are not needed, thus a single SBSML can include a combination of the semantic options of these semantic aspects.

Concurrency: A dichotomy in the semantics of BSMLs is whether a small step can include more than one transition or exactly one transition. Using the semantic option that a maximal set of transitions can be taken together in a small step along with a synchronizer UUEE(a), an *And* control state can be constrained to execute at most one of its transitions at each small step: Every transition within the *And* control state is modified to synchronize with a new self transition: t:{$\overline{a}$}.[3]

Preemption: Some BSMLs support *preemptive* semantics, in which an *interrupt transition* can supersede another transition, whose scope is lower than the interrupt transition. Other BSMLs support *non-preemptive* semantics, in which an interrupt transition can be taken together with the interrupted transition. A BSML that supports the non-preemptive semantics can model the preemptive semantics, by using synchronizers of type PUEE. For example, in the model in Fig. 3(a), transition st_4, which is an interrupt transition, can be taken together with transitions st_2 and st_3 by the non-preemptive semantics. Similarly, transition st_5 can be taken together with transitions st_1 and st_2. In the model in Fig. 3(b), which simulates the preemptive semantics, for each pair of transitions in which one is an interrupt and the other is interrupted, only one of them can be taken in a small step. For example, transitions dt_4 and dt_2 cannot be taken together because only one of them can synchronize with t_2.

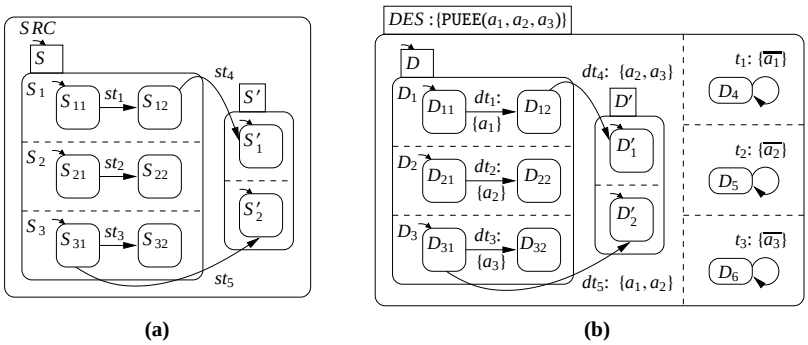


Fig. 3. Deriving preemptive behaviour in a non-preemptive semantics.

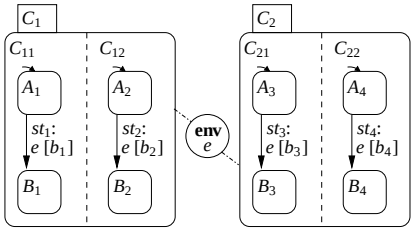
4.3 Modelling Composition Operators

In our previous work on template semantics [15], a set of composition operators were introduced, each of which represents a useful execution pattern in modelling. In this section, we describe how the behaviour of *rendezvous*, *environmental synchronization*, and *interleaving* composition operators can be modelled using synchronizers. For each of the remaining composition operators in template semantics, there is a similar *workflow pattern* [1], whose semantics we consider in the next section.

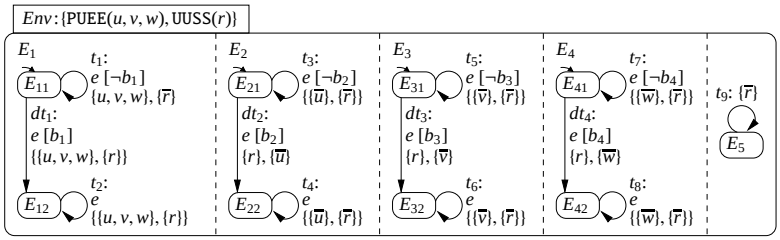
[3] This mapping is analogues to the derivation of *asynchrony* from *synchrony* in SCCS [13, 14].

Rendezvous: The rendezvous binary composition operator, analogous to the CCS composition operator [14], requires one of the transitions of one of its operands to generate a synchronization event and one of the transitions of the other operand that is triggered with that event to consume it, in the same small step. The semantics of the rendezvous composition operator is the synchronizer of type UUEE.

Environmental Synchronization: The environmental synchronization composition operator requires its two operands to synchronize over transitions that are triggered with the same synchronization event received from the environment. At each snapshot, it is possible that no, one, or more than one transition in each operand is enabled and triggered with the synchronization event. When all concurrent parts of the operands have enabled transitions that are triggered with the synchronization event, a synchronizer of type PUEE can be used to execute all of them together in the same small step. Otherwise, when there is an arbitrary number of enabled transitions that are triggered with the synchronization event, additionally a synchronizer of type UUSS is needed to ensure that a maximal set of such transitions are taken together in the same small step. As an example, the model in Fig. 4(a) uses the environmental synchronization composition operator over event e to coordinate the execution of C_1 and C_2. Each of the transitions in the model has a guard condition, enclosed by a "[ ]", on a boolean variable that is assigned a value by the environment. Fig. 4(b) is an SBSML model that has the same behaviour as the model in Fig. 4(a). Each control state of the model in Fig. 4(b) has a self transition to accommodate for the execution of synchronization transitions when not all of them are ready to execute. Self transition t_9 is necessary when executing dt_1, dt_2, dt_3, dt_4 together in the same small step. Transitions dt_1, dt_2, dt_3, dt_4 each synchronizes via two synchronizers.



(a) Environmental synchronization composition operator.



(b) Equivalent SBSML model for the model in Fig 4(a).

Fig. 4. Modelling the environment synchronization operator in SBSMLs.

Interleaving: The interleaving composition operator requires that exactly one of its operands to execute its transitions in each small step, exclusively. As an example, in the model in Fig. 5(a), in each small step, either transitions of C_1 or those of C_2 should be executed. The SBSML model in Fig. 5(b) uses a synchronizer of synchronization type UUSE to enforce the interleaving semantics of the model in Fig. 5(a), by executing either t_1 or t_2 in a small step, but not both.

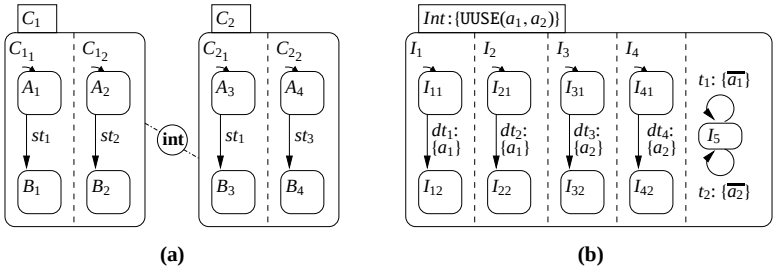
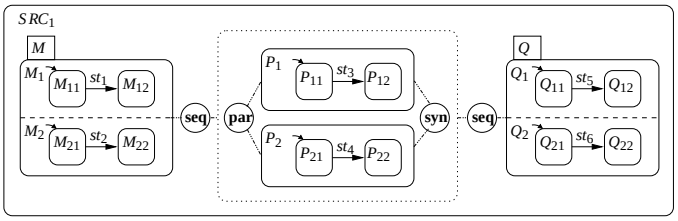


Fig. 5. Modelling the interleaving composition operator in SBSMLs.

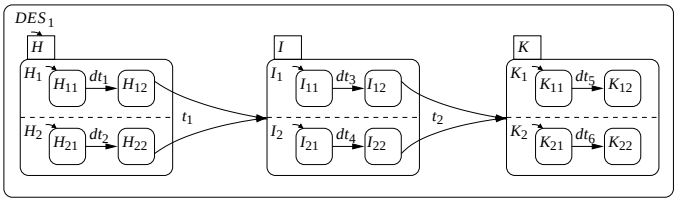
4.4 Modelling Workflow Patterns

There are five *basic workflow patterns* [1]: (i) *sequence*, which executes two activities sequentially: Once the first activity finishes, the second activity starts; (ii) *parallel split*, which starts executing multiple activities in parallel; (iii) *synchronization*, which merges multiple parallel activities into a single activity; (iv) *exclusive choice*, which non-deterministically chooses to execute one activity from a set of possible activities; and (v) *simple merge*, which requires exactly one of the alternative activities to finish before a next activity starts. Fig. 6 shows examples of how each of the basic workflow patterns can be modelled in SBSMLs. The model in Fig. 6(a) uses the first three patterns. The model in Fig. 6(c) uses the last two patterns, together with the sequence pattern. The circled operators **seq**, **par**, **syn**, **xor**, and **mer** represent the sequence, parallel split, synchronization, exclusive choice, and simple merge workflow patterns, respectively. Fig. 6(b) and Fig. 6(d), which use multi-source, multi-destination transitions, are equivalent SBSML models for the models in Fig. 6(a) and Fig. 6(c), respectively. The parallel split and synchronization workflow patterns are usually used together; e.g., as in the fork & join construct in activity diagrams in UML [16]. For the sake of brevity, we modelled only the basic workflow patterns, but it is not difficult to derive the semantics of other patterns, especially the ones that do not involve instances of activities.

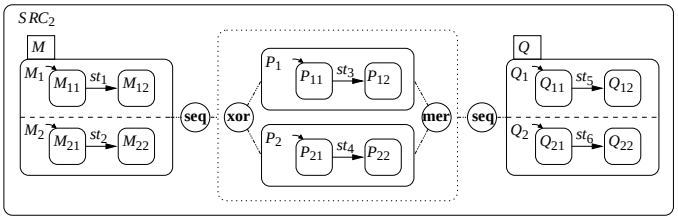
In interpreting and modelling the semantics of the sequence pattern above, an additional idle small step is introduced between the last small step of the first activity and the first small step of the second activity. This extra small step can be avoided by using an interrupt transition that transfers the control flow to the second activity simultaneously when the last small step of the first activity is being executed.



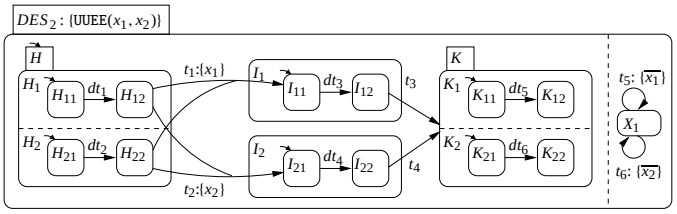
(a) A workflow model using sequence, parallel split, and synchronization workflow patterns.



(b) Equivalent SBSML model for the model in Fig. 6(a).



(c) A workflow model using sequence, exclusive choice, simple merge workflow patterns.



(d) Equivalent SBSML model for the model in Fig. 6(c).

Fig. 6. Modelling workflow patterns in SBSMLs.

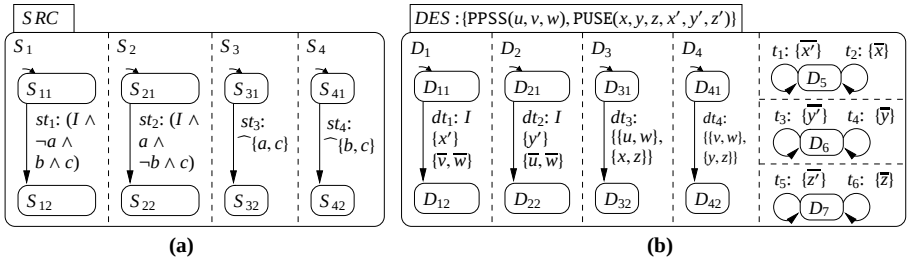


Fig. 7. Modelling the semantics of a notion of *signal* in SBSMLs.

4.5 Modelling Signals and Negations of Signals

Some BSMLs, such as μ-Charts [17], support a notion of a *signal* that when generated in a small step of a model can be sensed as present by all of the transitions of the model in the same small step.[4] The semantics of such signals can be modelled by synchronizers of type PPSS. A set of signals generated by a transition corresponds to a poly-role. The conjunction of signals in the trigger of a transition corresponds to a poly-co-role. To model the negation of a signal, a synchronizer of type PUSE and two labels can be used to disallow both a signal to be generated by a transition and its negation to be the trigger of a transition, in the same small step. As an example, in the model in Fig. 7(a), on page 12, when the model is initialized and input signal I is received from the environment, either of the potential small steps $\{st_1, st_4\}$ and $\{st_2, st_3\}$ can be taken, non-deterministically. (In the model in Fig. 7(a), the set of generated signals of a transition is prefixed with a "$\widehat{\ }$"). The model in Fig. 7(b) has the same behaviour as the one in Fig. 7(a). Labels u, v, and w correspond to signals a, b, and c, respectively. Input signal I is maintained in the model in Fig. 7(b). Labels x, y, z, together with labels x', y', z', ensure that each of the signals a, b, and c can be exclusively either generated or its negation can trigger a transition, respectively. For example, transitions dt_1 and dt_3 cannot be taken together because t_1 and t_2 cannot be taken together.

5 Related Work

Our classification of synchronization types overlaps with the classification of *multiparty interaction mechanisms* by Joung and Smolka [10]. They present a novel classification for synchronization mechanisms based on four criteria, which, in our terminology, can be described as: (i) whether or not the role sets and co-role sets of all transitions are singletons; (ii) whether or not a transition, in addition to specifying its role sets and co-role sets, can specify a particular transition (or transitions in a part of the hierarchy tree) with which it wishes to synchronize; (iii-a) whether or not the number of role sets and co-role sets of a transition together can be more than one; (iii-b) whether or not a control state can be the source control state of more than one transitions that can synchronize; and (iv) whether only a minimal set of synchronizing transitions are taken at each small step or a maximal set of all synchronizing transitions should be taken at each small step. Criterion (i) corresponds to the first two letters of our synchronization types, with our criteria being more distinguishing. Criterion (ii) is not relevant for us since it can be modelled by a naming convention for labels (cf., [10, p. 85]). Criterion (iii), called *conjunctive vs. disjunctive parallelism* [10], is meant to distinguish between process algebras such as SCCS (synchronous CCS) [13], which can perform multiple handshakes in one small step, and CCS, which can do at most one handshake; this criterion is closely related to the criterion (i) [10, p.83]. Part (a) of the criterion is not relevant in our framework since multiple role sets, or multiple co-role sets, related to the same synchronizer are merged into one. Part (b) of the criterion corresponds to a

[4] In previous work [5, 6], we have categorized this semantics of signals as the *same* semantic variation for *events*.

syntactic constraint in our framework. Lastly, we do not consider criterion (iv), focusing on semantics in which a maximal set of synchronizing transitions is always taken.

Compared to Joung and Smolka's taxonomy, our framework additionally considers the role of the third and fourth letters of our synchronization types. Also, additionally, our framework permits multiple synchronization types in one language. In general, the taxonomy of Joung and Smolka "is based on issues that may affect the complexity of scheduling multiparty interaction" [10, p.78], where as our framework is based on issues relevant for designing suitable modelling languages for requirements specification.

Our synchronizer syntax is inspired by the *encapsulation operator* in Argos [12]. The encapsulation operator specifies the scope in which a signal can be used. Our syntax is different in that multiple synchronizers can be attached to the same control state.

A class of BSMLs called synchronous languages [7], which includes languages such as Esterel [3] and Argos [12], have been successful in the specification of deterministic behaviour: "In contrast with traditional thread- or event-based concurrency models that embed no precise or deterministic scheduling policy in the design formalism, synchronous language semantics take care of all scheduling decisions." [20] The main characteristic of the synchronous languages is that the statuses of *signals* of a model are constant throughout a big step, which, in turn, introduces semantic difficulties such as non-causality and global inconsistency [3,7,12,18]. Using the synchronization capability of SBSMLs, it is possible to simulate the subset of the responsibilities of signals in synchronous languages that deal with the coordination of the simultaneous execution of transitions. As such, when using signal-like artifacts is not an option in a domain, e.g., UML StateMachines [16], synchronization could be used to achieve determinism in a model, by constructing the model such that each snapshot yields a unique small step.

SBSMLs, however, as opposed to synchronous languages, do not guarantee determinism as an inherent property of their semantics.[5] When a deterministic behaviour is desired in an SBSML model, care should be taken when using a synchronizer that has a synchronization type with its third and/or fourth letter being S, which permits synchronization with an arbitrary number of transitions. Similarly, care should be taken when using multiple synchronizers in a model, which could allow multiple sets of transitions to synchronize, according to different synchronizers, thereby creating different potential small steps. As an example, in the model in Fig. 7(b), if we remove labels x, y, and z from the model and replace x, y, and z in the co-role sets of t_2, t_4, t_6 with u, v, and w, respectively, the model can create a wrong small step that would include dt_1, dt_2, dt_3, and dt_4. The wrong small step is possible because label x' in dt_1 can match its corresponding label in t_1, while label u of dt_3 can match its corresponding label in dt_2. Similarly, dt_2 and dt_4 can match their corresponding labels in t_3 and dt_1, respectively.

6 Conclusion and Future Work

We presented a framework of 16 synchronization types based on criteria relevant for requirements modelling languages. We described how the class of big-step modelling

[5] A model in a synchronous language with a possible nondeterministic behaviour is conservatively rejected at compile time.

languages (BSMLs) can be enhanced with these synchronization types creating the family of synchronizing big-step modelling languages (SBSMLs). We validated the usefulness and generality of our framework by describing how underlying the semantics of many modelling constructs, there is a notion of synchronization that can be modelled in SBSMLs. Similar to our framework for BSMLs [4], we are working on a parametric framework to give formal semantics to the family of SBSMLs. Using the results of Joung and Smolka [10], we plan to analyze the complexity of implementing a set of synchronization types in an SBSML, to provide measures for a language designer or a software engineer to choose one SBSML over another. Lastly, we plan to provide a parametric tool support for SBSMLs, in the same style as in our previous work [11,19].

References

1. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
2. von der Beeck, M.: A comparison of Statecharts variants. In: FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 128–148. Springer (1994)
3. Berry, G., Gonthier, G.: The Esterel synchronous programming language: Design, semantics, implementation. Science Computer Programming 19(2), 87–152 (1992)
4. Esmaeilsabzali, S., Day, N.A.: Prescriptive semantics for big-step modelling languages. In: FASE 2010. LNCS, vol. 6013, pp. 158–172. Springer Verlag (2010)
5. Esmaeilsabzali, S., Day, N.A., Atlee, J.M., Niu, J.: Semantic criteria for choosing a language for big-step models. In: RE 2009. pp. 181–190. IEEE Computer Society Press (2009)
6. Esmaeilsabzali, S., Day, N.A., Atlee, J.M., Niu, J.: Deconstructing the semantics of big-step modelling languages. Requirements Engineering 15(2), 235–265 (2010)
7. Halbwachs, N.: Synchronous Programming of Reactive Systems. Kluwer (1993)
8. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)
9. Hoare, T.: Communicating Sequential Processes. Prentice Hall (1985)
10. Joung, Y.J., Smolka, S.A.: A comprehensive study of the complexity of multiparty interaction. Journal of the ACM 43(1), 75–115 (1996)
11. Lu, Y., Atlee, J.M., Day, N.A., Niu, J.: Mapping template semantics to SMV. In: ASE 2004. pp. 320–325 (2004)
12. Maraninchi, F., Rémond, Y.: Argos: an automaton-based synchronous language. Computer Languages 27(1/3), 61–92 (2001)
13. Milner, R.: Calculi for synchrony and asynchrony. Theoretical Computer Science 25(3), 267–310 (1983)
14. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
15. Niu, J., Atlee, J.M., Day, N.A.: Template semantics for model-based notations. IEEE TSE 29(10), 866–882 (2003)
16. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, v2.1.2 (2007), formal/2007-11-01
17. Philips, J., Scholz, P.: Compositional specification of embedded systems with statecharts. In: TAPSOFT 1997. LNCS, vol. 1214, pp. 637–651. Springer-Verlag (1997)
18. Pnueli, A., Shalev, M.: What is in a step: On the semantics of statecharts. In: TACS 1991. LNCS, vol. 526, pp. 244–264 (1991)
19. Prout, A., Atlee, J.M., Day, N.A., Shaker, P.: Semantically configurable code generation. In: MoDELS 2008. LNCS, vol. 5301, pp. 705–720 (2008)
20. Tardieu, O.: A deterministic logical semantics for pure Esterel. ACM TOPLAS 29(2), 8:1–8:26 (2007)