Theoretical Computer Science 429 (2012) 118–127

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Multi-modal scenarios revisited: A net-based representation

David Harel, Amir Kantor *

Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel

ARTICLE INFO

Keywords:
Live sequence charts
LSC
Scenario
Modal state structure
LSC net

ABSTRACT

Live sequence charts (LSC) is a visual formalism that can be used to model reactive systems. In terms of LSC, a system model is a set of charts, each of which may be referred to as a *multi-modal scenario*. In this paper, we revisit the scenarios of the UML2-compliant dialect of LSC. We abstract from their concrete visual representation, and show how to capture multi-modal scenarios in a flexible, yet conservative, way. For this, we use the building blocks of *Petri nets*; i.e., places and transitions, extended with modalities and interpreted in accordance with the semantics of LSC. This results in what we refer to as *LSC nets*. LSC nets can express a variety of advanced constructs of LSC, as well as several semantic variations suggested in the literature, with just a few primitive notions. At the same time, the net corresponding to an LSC involves rather superficial, technical, changes in presentation. As a result, LSC nets form a rigorous basis to present, discuss, and investigate the language as a whole, or interesting fragments thereof.

1. Introduction

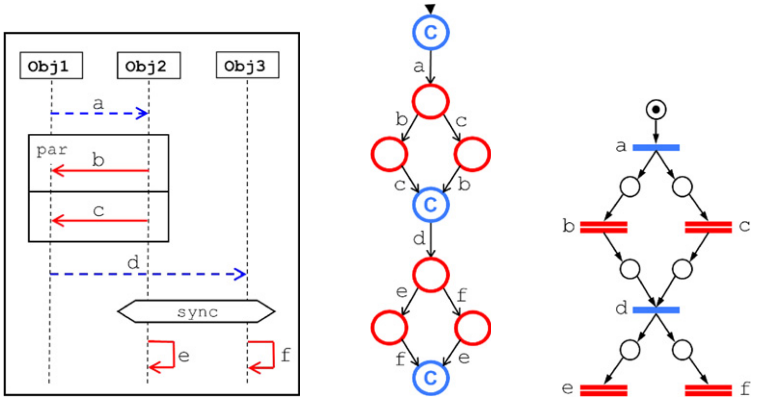
Live sequence charts (LSC) [4,8] is a visual formalism to model reactive systems through their inter-object interactions — namely, *scenarios*. The language extends classical *message sequence charts (MSC)* [9], mainly by being *multi-modal*; i.e., distinguishing between behaviors that *may* happen in the system (viz., *cold*, possible) and those that *must* happen (viz., *hot*, mandatory). With these two modalities, LSC can express a lot more; e.g., behaviors that are not allowed to happen (such as forbidden scenarios). LSC's scenario-based approach allows one to *define* the behavior of the modeled system through inter-object interactions, and does not force one to consider the separate behavior of each object. This is in contrast to the traditional intra-object approach, e.g., statecharts [5], where a reactive system is defined through the behavior of each of its objects.

LSC_A, which appears in Fig. 1a, is an example of a live sequence chart in a UML2-compliant variant of the language defined and discussed in [7].[1] In this paper, we mainly consider that dialect of the language, which is slightly generalized and more uniform than the original. LSC_A is a universal chart; i.e., an invariant that has to be true in any execution of the system. It captures the following multi-modal scenario. If and when `Obj1` sends message **a** to `Obj2` (a cold, possible, event; indicated by a dashed arrow in blue), the latter must reply with both **b** and **c** (hot, mandatory, events; indicated by solid arrows in red). The construct `par` designates that **b** and **c** have no ordering constraint; i.e., they are considered as if drawn in parallel. Then, if `Obj1` sends **d** to `Obj3`, `Obj2` must perform **e** (formally, a self-message), and, likewise, `Obj3` must perform **f**. The order of the events is determined by their attachment to the vertical lines in chart — the *lifelines* of the objects. However, the chart contains a synchronization point, denoted by `sync`, which requires that both **e** and **f** are to occur after **d**. Between **e** and **f**, in contrast, there is no ordering constraint.

* Corresponding author.
E-mail addresses: dharel@weizmann.ac.il (D. Harel), amir.kantor@weizmann.ac.il (A. Kantor).

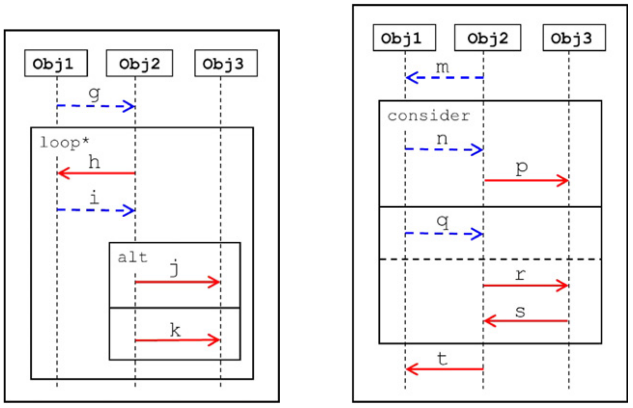
[1] In [7] this variant is called modal sequence diagrams (MSD); as in later papers we refer to it here simply as LSC.

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.12.030



(a) LSC_A. (b) MSS_A. (c) Net_A.

Fig. 1. Representing multi-modal scenarios.



(a) LSC_B. (b) LSC_C.

Fig. 2. Additional examples of live sequence charts.

LSC_A presents a few basic constructs of LSC. In order to model more complicated behavior, advanced constructs have been included in the language (see [7]). LSC_B of Fig. 2a includes a construct for an unbounded loop, annotated `loop*`, inside which appears a construct for alternatives, annotated `alt`. LSC_C, depicted in Fig. 2b, presents another construct, called 'consider'. It is the multi-modal generalization of the construct originally included in UML2, as discussed in [7] (we defer the description of LSC_B, LSC_C, and their underlying constructs to Section 4).

In the literature of LSC, e.g., [7,8], each construct like the ones in LSC_A, LSC_B, and LSC_C, is treated as a new primitive concept. It is therefore accompanied by corresponding definitions that augment the syntax and the semantics of the language. Here we abstract from the concrete visual representation of LSC, and show how to capture multi-modal scenarios in a high level of generality with just a few primitive notions. The idea is to incorporate the building blocks of *Petri nets* [14,13,15] into scenarios, and adapt them to the LSC setting. This results in a flexible net-based representation of the language, which we refer to in the following as *LSC nets*. It generalizes advanced constructs and semantic variations of LSC in a way that admits natural, direct, translations.

The paper is structured as following. A preliminary abstraction of multi-modal scenarios in the form of *modal state structures* [6] is briefly discussed in Section 2. We reach our main contribution in Section 3, in which LSC nets are introduced. In Section 4 we present translations of UML2-compliant LSCs into net form. The expressive power of LSC nets and their succinctness are explored in Section 5. Concluding remarks are included in Section 6.

2. Modal state structures

The semantics of an LSC may be presented via a translation into a Büchi automaton, as done in [11,7]. As an intermediate step, a process of 'unwinding' the chart results in a structure that is essentially a transition system with modalities.

This transition system, called a *modal state structure (MSS)* [6], captures the multi-modal scenario encoded in the chart. It is used here as a preliminary abstraction of scenarios. In contrast to LSC nets, which are presented in Section 3, this is a crude abstraction, which does not preserve the structure of the chart and may involve an exponential blowup. Nevertheless, it does capture the essential features of multi-modal scenarios. In the following we give a brief account of MSS. We later use MSSs in order to present the semantics of LSC nets in a rather clean way, to discuss their expressive power, and, in general, to expose their underlying concepts. A more elaborate account of MSS appears in [6].

2.1. Syntax and semantics of MSS

Fig. 1b depicts MSS_A, which captures the scenario in LSC_A of Fig. 1a. We regard the sending and the receiving of each message (drawn as a horizontal arrow in LSC_A) as a single event. Such an event is regarded as containing the information about its source and target, so that this information is not lost in MSS_A. Events appear as labels on the transitions of MSS_A. Each of the states of MSS_A corresponds to a point in the progress, from top to bottom, along the vertical lifelines of LSC_A; its initial state is drawn with a black triangle. The temperature of a state, cold or hot, indicates whether the state of the scenario is considered stable or not. Cold states, which are drawn in blue and marked with the letter 'C', are stable, while hot ones in red are not — the latter carrying a commitment to arrive sometime later at a cold state. We hereby interpret LSC_A in the *immediate* sense of [6]. This is a restrictive interpretation, where, roughly, any event appearing 'out of order' causes a violation of the scenario. Other, more permissive interpretations, are also available, and are easily represented both in MSS and in LSC nets; see Section 4.

Definition 1. A *modal state structure (MSS)*, $\mathcal{M}$, is defined as a tuple $\mathcal{M} = \langle \Sigma, S, s_0, \rightarrow, C \rangle$ where the following hold.

1. $\langle \Sigma, S, s_0, \rightarrow \rangle$ is a finite labeled transition system (LTS); i.e., Σ is a finite alphabet of *events*, S is a finite set of *states*, $s_0 \in S$ is the *initial state*, and $\rightarrow \subseteq S \times \Sigma \times S$ is a *labeled transition relation*.
2. $C \subseteq S$ are designated as *cold* states. All other states, namely, $H := S \setminus C$, are *hot*.

Let $\mathcal{M} = \langle \Sigma, S, s_0, \rightarrow, C \rangle$ be an MSS. For all $s, s' \in S$ and $e \in \Sigma$, we denote $s \xrightarrow{e} s'$ whenever $\langle s, e, s' \rangle \in \rightarrow$. For all $s \in S$ and $e \in \Sigma$, e is *enabled* in s, denoted $s \xrightarrow{e}$, whenever there exists $s' \in S$ such that $s \xrightarrow{e} s'$; i.e., e is on some transition outgoing from s. Given a state $s \in S$, $\text{E}(s) \triangleq \{e \in \Sigma : s \xrightarrow{e}\}$ denotes the set of the events enabled in s. We say that s is *dead-end* whenever $\text{E}(s) = \emptyset$. $\text{V}(s) \triangleq \Sigma \setminus \text{E}(s)$ denotes all other events, which are said to be *violating* in s.

MSSs are automata that adopt the notion of hot/cold modality of LSC, instead of traditional acceptance conditions. An MSS $\mathcal{M}$ is interpreted as following. An infinite word $\alpha \in \Sigma^\omega$ denotes a chain of events, and may designate a trace of an infinite execution. The *language* of $\mathcal{M}$, denoted $\mathfrak{L}(\mathcal{M})$, is the set of $\alpha \in \Sigma^\omega$ that are *accepted* by $\mathcal{M}$ (i.e., those that comply with $\mathcal{M}$). α is accepted by $\mathcal{M}$ if there is an *accepting run* of $\mathcal{M}$ on α, as follows.

An accepting run of $\mathcal{M}$ on α begins at the initial state. At each step $i \in \omega$, residing in some state s_i, an event α_i occurs. If α_i is enabled in s_i, i.e., $\alpha_i \in \text{E}(s_i)$, it must lead to a state s_{i+1} such that $s_i \xrightarrow{\alpha_i} s_{i+1}$ (there may be several possible successor states, each results in a different run). If, however, α_i is violating, i.e., $\alpha_i \in \text{V}(s_i)$, a *violation* occurs. Such a violation is hot or cold according to the temperature of s_i. A hot violation is regarded as illegal, since a hot state is unstable and carries a commitment to eventually reach a cold state, yielding a rejection of the run. Put differently, there are no hot violations in accepting runs. On the other hand, a cold violation, also called a *completion*, yields the acceptance of the run regardless of future events. If no violation occurs during the run, it is accepting iff every hot state is followed by a cold state; i.e., cold states occur infinitely often.

MSSs resemble Büchi automata on infinite words [16]. A Büchi automaton has two kinds of states: accepting and non-accepting. A run of the automaton on an infinite word is accepting iff accepting states are visited infinitely often. The difference between MSS and Büchi automata lies in the semantics of disabled events. In MSS, if a disabled event occurs in state s, it yields the acceptance or the rejection of the run depending on the temperature of s. In Büchi automata, in contrast, disabled events are always rejected, in both accepting and non-accepting states. That kind of duality of hot and cold states lies at the heart of multi-modal scenarios of LSC.

One way to formally express the intended semantics of MSS is to translate $\mathcal{M}$ it into an equivalent Büchi automaton, denoted $\mathcal{B}_\mathcal{M}$. The rules of the translation are illustrated in Fig. 3 and are given by:

Definition 2. The Büchi automaton that corresponds to $\mathcal{M}$ is defined by $\mathcal{B}_\mathcal{M} \triangleq \langle \Sigma, \overline{S}, s_0, \Delta, F \rangle$, where Σ is the alphabet of $\mathcal{B}_\mathcal{M}$, $\overline{S} := S \dot{\cup} \{\top\}$ are its states, and s_0 is the initial state. $F := C \cup \{\top\}$ is the set of accepting states of $\mathcal{B}_\mathcal{M}$, and its transition relation Δ is given by

$$\Delta := \rightarrow \ \cup \{ \langle s, e, \top \rangle : s \in C,\ e \in \text{V}(s) \} \ \cup \ \{ \langle \top, e, \top \rangle : e \in \Sigma \}.$$

The language of $\mathcal{M}$ is defined accordingly by $\mathfrak{L}(\mathcal{M}) \triangleq \mathfrak{L}(\mathcal{B}_\mathcal{M}) \subseteq \Sigma^\omega$.

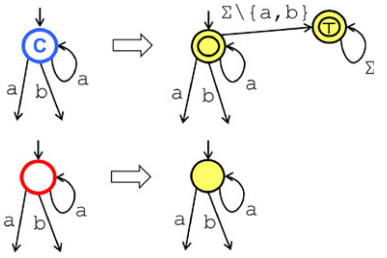


Fig. 3. Translation of an MSS into a Büchi automaton.

2.2. Expressive power of MSS

Given a Büchi automaton $\mathcal{B}$ over alphabet Σ, we can easily translate it into an equivalent MSS $\mathcal{M}_{\mathcal{B}}$ over Σ (see [6]). We therefore obtain the following.

Proposition 1. *Given a finite alphabet Σ, MSSs over Σ are just as expressive as Büchi automata over Σ; i.e., they yield the ω-regular languages.*

The basic interpretation suggested above for an MSS $\mathcal{M}$ — that is, the language $\mathfrak{L}(\mathcal{M})$ — is a collection of those chains of events that comply with the scenario from their beginning. This language corresponds directly to the *initial* semantics of $\mathcal{M}$ (cf. [3,12]). As in the case of LSCs, it may be used in order to define other semantics for $\mathcal{M}$, such as a *universal* semantics (invariance) and an *existential* semantics (a positive example); see, e.g., [6].

3. LSC nets

In MSS, one needs to specify the transitions of each state separately. In many situations, however, taking a single transition should cause only a *partial* change in the required behavior. Consider for instance LSC_A of Fig. 1a after **d** occurs. There are two events that must take place, **e** and **f**. After **e** occurs, **f** is still required to occur. The required behavior changed, but only partially. In MSS this needs to be specified in each state separately; see Fig. 1b. The independence of states in MSS yields additional expressive power, as we have seen, but results in a fundamental change to the structure of scenarios and their succinctness.

In classical Petri nets [14,13,15] transitions cause only partial, local, change in the global state. This principle traditionally suggests interpretations intended to capture such notions as concurrency and distribution. In the following we exploit the power of nets to represent multi-modal scenarios. This results in an expressive framework of *LSC nets*. With a small number of primitive notions, LSC nets allow one to encode general patterns of behavior in a form that is much closer to LSC.

Petri nets. We now present a brief introduction to Petri nets (more specifically, to place/transition nets), presenting basic terminology that is later used in our definition of LSC nets. For a more elaborate account of Petri nets, the interested reader is referred to, e.g., [14,13].

Place/transition nets (P/T-nets, or PTN) are an abstract model for the flow of control and information in systems, particularly distributed and concurrent systems. A P/T-net $\mathcal{C}$ is a directed bipartite graph, with nodes partitioned into a set of *places* P and a set of *transitions* T. Places are drawn as circles and transitions as bars, with *arcs* connecting them (see Fig. 1c). The global state of the net is given by a *marking*, a function μ that assigns each place $p \in P$ a nonnegative number of *tokens*, $\mu(p) \in \omega$. The set of all markings is denoted $\omega^P = \{\mu : P \to \omega\}$. The P/T-nets that we consider are marked; i.e., $\mathcal{C}$ is also given an *initial marking* drawn by assigning tokens to places. More formally, a P/T-net $\mathcal{C}$ is defined as a tuple $\mathcal{C} = \langle P, T, F, \mu_0\rangle$, where P is a finite set of places, T is a finite set of transitions ($P \cap T = \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation (arcs). $\mu_0 \in \omega^P$ is the initial marking.

Let $\mathcal{C} = \langle P, T, F, \mu_0\rangle$ be a P/T-net. For each transition $t \in T$, we denote by ${}^\bullet t \triangleq \{p \in P : \langle p, t\rangle \in F\}$ its *preset* of places, and by $t^\bullet \triangleq \{p \in P : \langle t, p\rangle \in F\}$ its *postset* of places. Similarly for places $p \in P$, ${}^\bullet p \triangleq \{t \in T : \langle t, p\rangle \in F\}$ and $p^\bullet \triangleq \{t \in T : \langle p, t\rangle \in F\}$.

Given a particular marking $\mu \in \omega^P$, a transition $t \in T$ is *enabled* in μ if each place in its preset has a token. If t is enabled, it may *fire*, which results in a new marking obtained by removing a token from each preset place, and adding a token to each postset place. More formally, we say that t is enabled in μ, denoted $\mu[t\rangle$, whenever for each $p \in {}^\bullet t$, $\mu(p) \geq 1$. When this holds, define $\mu^{[t]} \in \omega^P$, the result of firing t in μ, by

$$\mu^{[t]}(p) \triangleq \begin{cases} \mu(p) - 1 & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ \mu(p) + 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ \mu(p) & \text{if } p \in {}^\bullet t \cap t^\bullet \text{ or } p \in P \setminus ({}^\bullet t \cup t^\bullet). \end{cases}$$

For all $\mu, \mu' \in \omega^P$ and $t \in T$, we denote $\mu[t\rangle\mu'$ whenever $\mu[t\rangle$ and $\mu' = \mu^{[t]}$.

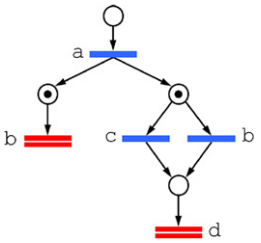


Fig. 4. Two enabled transitions labeled **b**.

3.1. Syntax and semantics of LSC nets

In order to obtain a net-based representation of multi-modal scenarios we now extend Petri nets with temperature. In analogy to MSS, each *marking* will be cold (stable) or hot (unstable). This can be done in a number of ways. First, we may consider various variants of Petri nets — with or without arc weights, capacity constraints on places, inhibitor arcs (zero-testing), etc. Guided by a desire for simplicity we present LSC nets based on basic P/T-nets. Second, the temperature of markings may be obtained by attaching the hot/cold modality to places, to transitions, or to both (similarly to defining the temperature of cuts in LSCs, cf. [8]). In order to adhere to the UML2-compliant variant of the LSC language [7], we attach temperature modality to transitions, and label transitions with events. Into this setting we can define notions related to temperature, such as hot and cold violations.

Fig. 1c shows Net_A, which corresponds to LSC_A of Fig. 1a. Transitions of Net_A, labeled with events, correspond to the events appearing in LSC_A. The temperature of a transition, cold or hot, is obtained from the temperature of the corresponding event in LSC_A. A hot transition is drawn as a double bar in red. Transitions and places in Net_A are structured according to the partial ordering of the events in LSC_A.

Definition 3. An *LSC net* $\mathcal{N}$ is defined as a tuple $\mathcal{N} = \langle \Sigma, P, T, F, l, C, \mu_0 \rangle$, where the following hold.

1. $\langle P, T, F, \mu_0 \rangle$ is a P/T-net.
2. Σ is a finite alphabet of events, and $l : T \to \Sigma$ is a *transition-labeling function*.
3. $C \subseteq T$ is the set of *cold* transitions. All other transitions, namely, those in $H := T \setminus C$, are *hot*.

Let $\mathcal{N} = \langle \Sigma, P, T, F, l, C, \mu_0 \rangle$ be an LSC net. From the underlying P/T-net of $\mathcal{N}$, namely $\langle P, T, F, \mu_0 \rangle$, notions such as markings and enabled transitions are already defined. $\mathcal{N}$, however, represents a multi-modal scenario in a way quite similar to an MSS, which is emphasized here by choosing similar phrases (cf. Section 2.1). The semantics is presented formally later, through a translation of an LSC net into a corresponding MSS. The *language* of $\mathcal{N}$, denoted $\mathfrak{L}(\mathcal{N})$, is the set of $\alpha \in \Sigma^\omega$ that are *accepted* by $\mathcal{N}$ (i.e., those that comply with $\mathcal{N}$). α is accepted by $\mathcal{N}$ if there is an *accepting run* of $\mathcal{N}$ on α, as follows.

A marking is *hot* whenever it contains an enabled hot transition, otherwise it is *cold*. A cold marking is stable, whereas a hot marking is not. In each marking there is a set of *enabled events* — the labels of enabled transitions — while all other events are considered *violating*. An accepting run of $\mathcal{N}$ on α begins with the initial marking. At each step $i \in \omega$, residing in some marking μ_i, an event α_i occurs. If α_i is enabled in μ_i, a corresponding transition is fired (see details below), which leads to a new marking μ_{i+1}. If, however, α_i is violating, a *violation* occurs. Such a violation is hot or cold according to the temperature of μ_i. A hot violation is regarded as illegal, since a hot marking is unstable and carries a commitment to eventually reach a cold marking, yielding a rejection of the run. Put differently, there are no hot violations in accepting runs. On the other hand, a cold violation, also called a *completion*, yields the acceptance of the run regardless of future events. If no violation occurs during the run, it is accepting iff every hot marking is followed by a cold one; i.e., cold markings occur infinitely often.

Consider an event e enabled in a marking μ at some step during the run, such that there is more than one enabled transition labeled e (see, e.g., the net in Fig. 4, which has two enabled transitions labeled **b**). There are several possibilities as to which transition, or transitions, should fire if e occurs. Given an LSC specification, a system event may correspond to events enabled in *several*, distinct, scenarios. Similarly within a single scenario, when considering advanced constructs like `consider` and `par`, a system event can be recognized simultaneously in more than one place. Our objective is to faithfully represent multi-modal scenarios in net form, so the semantics of LSC nets must reflect this phenomenon. This is achieved through a *unification principle*, according to which the firings of *several* enabled transitions are identified as a single firing. For instance, in Fig. 4, if **b** occurs two transitions labeled **b** fire, as both are enabled. It is a fundamental difference from ordinary Petri net semantics, reflecting the declarative nature of multi-modal scenarios. Generally, when an enabled event e occurs, a *maximal non-conflicting* set of enabled transitions labeled e is fired. Accordingly, in the following we define the enabling of a *set* of transitions and its firing. We interpret 'non-conflicting' in a structural, static, sense. Another possibility is to consider a generally more inclusive, dynamic, interpretation (depending upon the current marking). On net-based representations of LSCs (see Section 4) both interpretations are equivalent, so we choose the one that seems simpler.

Definition 4. Given a marking $\mu \in \omega^P$ and $U \subseteq T$ a set of transitions,

1. U is *μ-established* whenever it is a nonempty set of enabled non-conflicting transitions that are labeled by the same event. That is, $U \neq \emptyset$, for each $t \in U$ holds $\mu[t\rangle$, and for all $t_1 \neq t_2 \in U$, ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$ and $l(t_1) = l(t_2)$.
2. U is *enabled* in μ, denoted $\mu[U\rangle$, if it is a maximal μ-established set of transitions. That is, U is μ-established, and there is no μ-established $V \subseteq T$ such that $U \subsetneq V$. When this holds define $\mu^{[U]} \in \omega^P$, the result of *firing* U in μ, by

$$\mu^{[U]}(p) \triangleq \mu(p) - |p^\bullet \cap U| + |{}^\bullet p \cap U|.$$

   This is equivalent to a sequence of firings of the transitions in U in an arbitrary order.
3. For all $\mu, \mu' \in \omega^P$ and $U \subseteq T$, $\mu[U\rangle\mu'$ whenever $\mu[U\rangle$ and $\mu' = \mu^{[U]}$.

We define the semantics of $\mathcal{N}$ by translating it into a corresponding MSS, denoted $\mathcal{M}_\mathcal{N}$. The alphabet of $\mathcal{M}_\mathcal{N}$ is the set of events Σ, its states are the set of markings ω^P, and μ_0 is its initial state. The temperature of a state is taken to be its temperature as a marking. Given markings $\mu, \mu' \in \omega^P$ and an event $e \in \Sigma$, there is a transition $\mu \xrightarrow{e} \mu'$ in $\mathcal{M}_\mathcal{N}$ iff there exists $U \subseteq T$ enabled in μ, where the transitions in U are labeled e, and μ' is the result of firing U in μ. The accepting runs of $\mathcal{N}$, and its language, are those of the corresponding $\mathcal{M}_\mathcal{N}$.

$\mathcal{M}_\mathcal{N}$ has, however, an infinite number of states. This is not a problem, since in the definition of an MSS one need not assume a finite number of states (i.e., one can ignore this assumption in Definition 1). In order to provide semantics for such a generalized MSS $\mathcal{M}$, we can still construct $\mathcal{B}_\mathcal{M}$ and ignore the assumption of finiteness of the set of states in Büchi automata. Under this modification, generalized MSSs may be viewed as a generic model — not so much as a concrete language — since an MSS with an infinite number of states does not come endowed with a finite representation. Particularly, the prescribed translation of an LSC net $\mathcal{N}$ into $\mathcal{M}_\mathcal{N}$ is not effective. Nevertheless, generalized MSSs can be used to define the semantics of LSC nets, whereas the latter have finite representations. In Section 5 we discuss an effective translation of bounded nets into MSSs.

Definition 5. The (generalized) MSS that corresponds to $\mathcal{N}$ is defined by $\mathcal{M}_\mathcal{N} \triangleq \langle \Sigma, \omega^P, \mu_0, \rightarrow, \widetilde{C}\rangle$, where $\widetilde{C} := \{\, \mu \in \omega^P : \nexists t \in H \text{ s.t. } \mu[t\rangle \,\}$ is the set of cold markings, and the transition relation is given by

$$\rightarrow := \{\, \langle \mu, e, \mu' \rangle \in \omega^P \times \Sigma \times \omega^P : \exists U \subseteq T \text{ s.t. } \mu[U\rangle\mu' \text{ and } \forall t \in U,\ l(t) = e \,\}.$$

The language of $\mathcal{N}$ is defined accordingly by $\mathfrak{L}(\mathcal{N}) \triangleq \mathfrak{L}(\mathcal{M}_\mathcal{N}) \subseteq \Sigma^\omega$.

Because $\mathcal{N}$ is essentially a form of an MSS, any available interpretation of MSSs, such as universal and existential semantics, is also defined for $\mathcal{N}$.

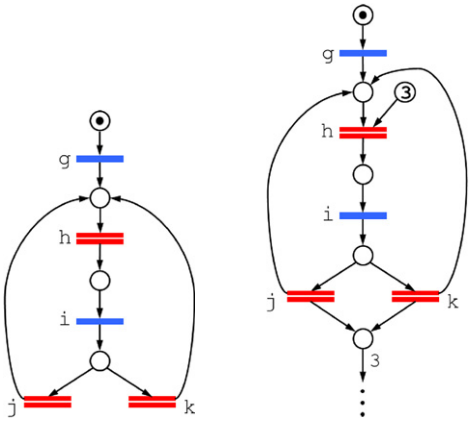
4. Translating LSCs into LSC nets

Constructs `sync` and `par`. LSC nets are a powerful means to capture multi-modal scenarios. We have already seen how the scenario of LSC_A, depicted in Fig. 1a, can be described using the equivalent Net_A in Fig. 1c. LSC_A involves the constructs `par` [2] and `sync` [8] whose sole purpose is to adjust the partial ordering of events in the chart.

Constructs `alt` and `loop`. LSC_B of Fig. 2a involves more advanced constructs. It prescribes that if and when **g** occurs, the rest of the scenario is preformed in a loop: **h** must take place, after which if **i** occurs then either **j** or **k**, which appear inside the construct `alt`, must take place. Construct `alt` [7,8,2] prescribes a nondeterministic alternative. Construct `loop*` [7,8] is an unbounded loop, which may be escaped through cold violations (i.e., completions). These patterns of behavior can be easily captured with LSC nets. The equivalent of LSC_B in terms of nets is Net_B, which appears in Fig. 5a.

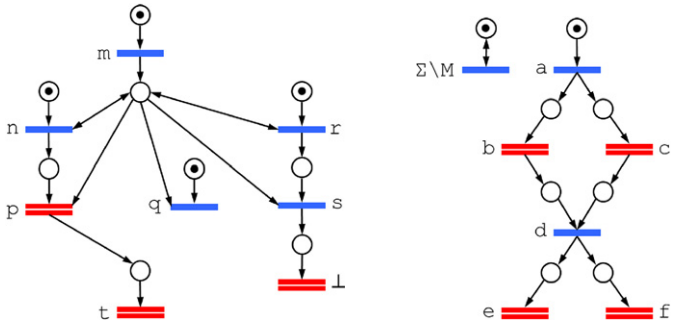
LSC also admits a construct for bounded loops [7,8]. Assume that the loop in LSC_B is turned into one that is repeated at most 3 times (in that case the construct would be annotated `loop 3`). Such a scenario can be described as an LSC net, $\text{Net}_{B'}$ of Fig. 5b. In contrast to an unbounded loop, a bounded one may be followed by events that come into play if and when the scenario continues. In order to capture this concisely as an LSC net, we slightly generalize the definition of nets to include weighted arcs (cf. [14]), which carry more than one token. This straightforward generalization is aimed to allow the last arc of $\text{LSC}_{B'}$ which consumes 3 tokens; it is included in order to illustrate the translation in the presence of events following the bounded loop.

Multi-modal `consider` construct. Another construct of LSC, termed `consider`, appears in LSC_C of Fig. 2b. The construct is introduced into multi-modal scenarios in [7]. It is comprised of a main operand, containing a fragment of a scenario, and several other, secondary, operands that are drawn below it. Secondary operands, which are either hot or cold, are to be considered concurrently to the main operand. When a secondary operand is completed, it results in a violation of the chart, which is either hot or cold depending on the temperature of the operand. In LSC_C, the `consider` construct is enabled if and when event **m** occurs. It first says that if event **n** occurs then **p** must take place. During that fragment of a scenario event **q** may occur, in which case a cold violation of LSC_C occurs (as the operand is cold), resulting in a successful completion of the



(a) Net_B. (b) $\text{Net}_{B'}$.

Fig. 5. Net-based representations of LSCs.



(a) Net_C. (b) $\text{Net}_{A'}$.

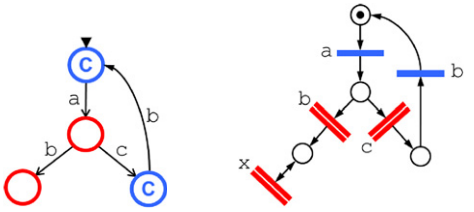
Fig. 6. More net-based representations of LSCs.

chart. Moreover, **r** may occur, but it cannot be followed by **s**, for that would cause a hot, illegal, violation (since this operand is hot). Event **t** must occur if and when the main operand is completed without violations.

This pattern of behavior may be captured in the equivalent Net_C depicted in Fig. 6a (bidirectional arrows stand for two arcs, one in each direction). In Net_C, each of the three operands is drawn in parallel, but their transitions synchronize on a common place. Maximal events in each operand consume the token from that common place so that the transitions of the `consider` operator become disabled. The symbol $\perp$ in Fig. 6a denotes a dummy event that never occurs, so that if the transition labeled **s** fires the run is rejected (instead of using a dummy event we can equivalently transform its underlying transition into a self-loop and label it by any other event). This translation is valid due to the fact that in LSC_C the events in the operands are disjoint (and in each operand there are no identical events in parallel), so unification cannot occur. In general one needs several common places for synchronization.

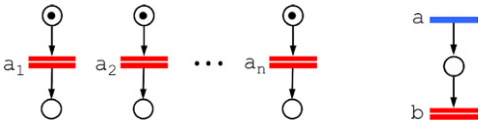
Restricted events, and the constructs `consider` *and* `ignore`. An LSC may be interpreted in several ways with respect to its set of *restricted* events; i.e., events that cause a violation if they occur 'out of order' (see [6]). Several particular interpretations of LSC have been suggested with respect to this set of restricted events [11,8,2]. When all events are restricted we call the chart *immediate*, and if no events are restricted, so that there are no violations, the chart is *tolerant*, or *weak*. If, however, the restricted events are exactly those appearing in the chart, the chart is called *strict*. The constructs `consider` and `ignore` can be used to adjust the set of restricted events (see [7]).

All these semantic variants can be easily represented with LSC nets. Net_A of Fig. 1c corresponds to the immediate interpretation of LSC_A. Note that MSS_A of Fig. 1b is the translation of Net_A into MSS (removing unreachable markings). A net-based representation of the strict interpretation of LSC_A appears in Fig. 6b (where $M := \{\mathbf{a}, \mathbf{b}, \ldots, \mathbf{f}\}$). Its translation into MSS may be obtained from MSS_A by adding to all states self-transitions labeled $\Sigma \setminus M$. A tolerant interpretation, as well as an interpretation set up to correspond to any set of restricted events, may also be represented as an LSC net. As in Fig. 6b, one adds a self-loop consisting of an additional place and additional cold transitions labeled with the events that are not restricted. The unification principle of LSC nets ensures the adequacy of the translation.



(a) MSS_D. (b) Net_D.

Fig. 7. Translating an MSS into an LSC net.



(a) Net_{E_n}. (b) Net_F.

Fig. 8. LSC nets.

We have illustrated the translation of LSCs and several of their constructs into LSC nets. A formal algorithm giving the full translation is outside the scope of this paper, mainly due to the fact that the nontrivial parts involve advanced constructs that have not yet been given formal semantics in the context of the UML2-compliant dialect of LSC [7].

5. On the expressive power and succinctness of LSC nets

The *reachable markings* of a P/T-net $\mathcal{C}$ are those that may be reached by a finite sequence of transition firings, starting from the initial marking. $\mathcal{C}$ is *bounded* if it has only a finite number of reachable markings. Boundedness may be decided effectively by constructing a finite *reachability tree* [13].

Now consider an LSC net $\mathcal{N}$. Any firing of an enabled set of transitions in $\mathcal{N}$ is a sequence of firings of individual transitions, so that any reachable marking as an LSC net, with the unification principle, is reachable in the traditional sense. Therefore, if $\mathcal{N}$ is bounded in the traditional sense, it also has a finite number of reachable markings as an LSC net. Such a net may be translated effectively into an MSS with only a finite number of states, by considering only reachable markings.

Given an MSS $\mathcal{M}$ over alphabet Σ, with a finite number of states, we can easily translate it into an LSC net over Σ, which will be denoted $\mathcal{N}_{\mathcal{M}}$. For example, MSS_D of Fig. 7a is translated into Net_D of Fig. 7b. We turn states into places, and labeled transitions of $\mathcal{M}$ into labeled transitions of $\mathcal{N}_{\mathcal{M}}$, each of which has a single preset place and a single postset place. The temperature of a transition is chosen according to the temperature of its source state in $\mathcal{M}$. For a dead-end hot state of $\mathcal{M}$ (see Fig. 7a), we add to the corresponding place in $\mathcal{N}_{\mathcal{M}}$ a hot transition in a self-loop (labeled with an arbitrary event), so that a run reaching that point is rejected. The initial marking has one token put in the place corresponding to the initial state of $\mathcal{M}$. The nets we obtain from this translation are bounded in the traditional sense. In fact, any reachable marking has exactly one token. From this we obtain the following:

Proposition 2. *Given a finite alphabet Σ, the set of bounded LSC nets over Σ is as expressive as the set of MSSs over Σ with a finite number of states. Both yield exactly the ω-regular languages.*

Although bounded LSC nets are not more expressive than MSSs, they are exponentially more succinct. This can be proved by the sequence of nets $\{\mathrm{Net}_{E_n}\}_{n=1}^{\infty}$. The element Net_{E_n} is shown in Fig. 8a; it is defined over the alphabet $\Sigma_n = \{\mathbf{a}_1, \ldots, \mathbf{a}_n, \mathbf{b}\}$ of events. The language of Net_{E_n} is $\mathfrak{L}(\mathrm{Net}_{E_n}) = P_n \cdot \Sigma_n^{\omega}$, where P_n is the set of all permutations of the events $\mathbf{a}_1, \ldots, \mathbf{a}_n$. It is not difficult to prove that in any Büchi automaton accepting $\mathfrak{L}(\mathrm{Net}_{E_n})$ there are at least 2^n states. For any subset $A \subseteq \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$, consider some permutation $\pi_A \in P_n$ beginning with the events in A, and take an accepting run ρ_A of the automaton on the word $\pi_A \mathbf{b}^{\omega} \in \mathfrak{L}(\mathrm{Net}_{E_n})$. Denote the state of ρ_A after reading the events in A by s_A. It follows that for any $A_1 \neq A_2 \subseteq \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$, we must have $s_{A_1} \neq s_{A_2}$, because otherwise we would have had an accepting run on a word not in $\mathfrak{L}(\mathrm{Net}_{E_n})$ as following. Assume w.l.o.g that $\mathbf{a}_i \in A_1 \setminus A_2$. Start advancing according to ρ_{A_2}, and, after the events in A_2 (while reaching $s_{A_2} = s_{A_1}$) continue according to ρ_{A_1}. As a result we get an accepting run on a word not containing $\mathbf{a}_i$. Because an MSS can be translated into a Büchi automaton having one additional state, we obtain the following.

Proposition 3. *Bounded LSC nets are exponentially more succinct than MSSs.*

Many important nets are bounded; e.g., condition/event systems [14], as well as all the LSC nets appearing in the examples throughout the paper (with the exception of Fig. 8b). As for the expressive power of general LSC nets, consider Net_F of Fig. 8b. It is an unbounded net, allowing tokens to accumulate at the intermediate place. Denote its language by $\mathfrak{L} := \mathfrak{L}(\mathrm{Net}_F)$. For any $j \in \omega$, the word $\mathfrak{a}^j\mathfrak{b}^j\mathfrak{c}^\omega \in \mathfrak{L}$, but for any $k < j$ the word $\mathfrak{a}^j\mathfrak{b}^k\mathfrak{c}^\omega \notin \mathfrak{L}$. Therefore, the language $\mathfrak{L}$ is not obtainable by a Büchi automaton. The standard way of proving such things works here too. Assume given a Büchi automaton $\mathcal{B}$ that accepts $\mathfrak{L}$. For any $j \in \omega$, denote by ρ_j an accepting run of $\mathcal{B}$ on $\mathfrak{a}^j\mathfrak{b}^j\mathfrak{c}^\omega$, and denote the corresponding state after reading the prefix $\mathfrak{a}^j$ by s_j. However, for any $k < j \in \omega$, $s_k \neq s_j$, since otherwise an accepting run on $\mathfrak{a}^j\mathfrak{b}^k\mathfrak{c}^\omega \notin \mathfrak{L}$ could have been constructed by advancing according to ρ_j on $\mathfrak{a}^j$ up until $s_j = s_k$, and then continuing from there according to ρ_k on $\mathfrak{b}^k\mathfrak{c}^\omega$. Thus, $\mathcal{B}$ would have an infinite number of states, which is a contradiction. The following thus follows:

Proposition 4. *LSC nets are strictly more expressive than MSSs with a finite number of states; i.e., they yield strictly more than the ω-regular languages.*

Two previous pieces of work discuss transformations of LSC specifications into (coloured) Petri nets [1,10]. Although the subject resembles that of the present paper, there are significant differences. First, both [1] and [10] consider LSC specifications in the dialect of [8], according to which an LSC is partitioned into a prechart and a main chart. Second, the form of the corresponding net, which aims to capture the configurations in which an LSC specification may reside during the progress of an execution, is generally very different from the original specification. Third, the semantics underlying LSC specifications is not preserved by those transformations. Specifically, the distinction between prechart events (which only need to be monitored) and main chart events (that need to be executed) is not expressed in the corresponding net. Moreover, as far as we see, the net does not adequately capture completions of charts due to cold violations.

6. Conclusion

LSC nets extend the dynamics of multi-modal scenarios, specifically those of the UML2-compliant variant of LSC [7], in a fundamental way. This is called for by some of the advanced constructs of LSC, which address the need to break the partial order structure of the chart to be able to encode more complicated behavior. The result is a flexible representation of scenarios.

As presented in Section 4, LSC nets can express UML2-compliant LSCs that include a variety of advanced constructs. The original LSCs are more structured, allowing the nesting of various constructs, each of which incorporates a certain pattern of behavior. However, such LSCs involve quite a few primitive notions, and each of them needs to be assigned syntax and semantics, so that they become conceptually and technically more complicated. The corresponding nets replace LSC's advanced constructs with enriched dynamics that characterizes their behavior in net form.

This comes at a price, since nets, in general, can become rather complicated. However, according to the scenario-based approach to system modeling, different concerns are separated into distinct scenarios, so that each scenario alone is typically not very complex. Moreover, the two approaches can be combined. LSC nets can be introduced as a formal framework for multi-modal scenarios, within which concrete visual syntax, advanced constructs, and semantic variations can be defined as abbreviations. The abstraction level of nets seems a natural way to discuss and investigate these notions.

Due to the notorious trade-off between expressive power and efficient algorithms, the full scope of LSC nets as presented thus far may be more than one would want for some applications. We saw that unbounded nets offer additional expressive power, so it may be beneficial to consider only bounded LSC nets, and stay in the realm of ω-regular languages. One way to limit the formalism in that spirit is to introduce a capacity constraint for places, requiring that each place holds no more than k tokens (cf. [14]). This adjustment yields nets with a finite number of markings, which still capture the full scope of the advanced constructs considered in this paper.

Acknowledgements

The authors wish to thank Shahar Maoz and Itai Segall for their helpful comments on the material in this paper. The research was supported in part by the John von Neumann Minerva Center for the Development of Reactive Systems at the Weizmann Institute of Science, and by an advanced research grant to the first-listed author, from the European Research Council (ERC), under the European Community's Seventh Framework Programme (FP7/2007-2013).

References

[1] L. Amorim, P.R.M. Maciel, M.N. Nogueira Jr, R.S. Barreto, E. Tavares, Mapping live sequence chart to coloured Petri nets for analysis and verification of embedded systems, ACM SIGSOFT Software Engineering Notes 31 (3) (2006) 1–25.

[2] Y. Bontemps, Relating Inter-Agent and Intra-Agent Specifications, Ph.D. Thesis, University of Namur, Computer Science Dept, April 2005.

[3] M. Brill, W. Damm, J. Klose, B. Westphal, H. Wittke, Live sequence charts: an introduction to lines, arrows, and strange boxes in the context of formal verification, in: H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, E. Westkämper (Eds.), SoftSpez Final Report, in: LNCS, vol. 3147, Springer, 2004, pp. 374–399.

[4] W. Damm, D. Harel, LSCs: breathing life into message sequence charts, Formal Methods in System Design 19 (1) (2001) 45–80.
[5] D. Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming 8 (3) (1987) 231–274.
[6] D. Harel, A. Kantor, Modal scenarios as automata, in: N. Dershowitz, E. Nissan (Eds.), Language, Culture, Computation, in: LNCS, vol. 1, Springer-Verlag, Berlin (in press).
[7] D. Harel, S. Maoz, Assert and negate revisited: modal semantics for UML sequence diagrams, Software and Systems Modeling (SoSyM) 7 (2) (2008) 237–252.
[8] D. Harel, R. Marelly, Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine, Springer-Verlag, 2003.
[9] ITU, International Telecommunication Union Recommendation Z.120: Message Sequence Charts, Technical report, 1996.
[10] B. Khadka, B. Mikolajczak, Transformation from live sequence charts to colored Petri nets, in: Proc. of the 2007 Summer Computer Simulation Conf., SCSC, pp 673–680, San Diego, CA, USA, 2007. Society for Computer Simulation International.
[11] J. Klose, H. Wittke, An automata based interpretation of live sequence charts, in: T. Margaria, W. Yi (Eds.), Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01), in: LNCS, vol. 2031, Springer, 2001.
[12] H. Kugler, D. Harel, A. Pnueli, Y. Lu, Y. Bontemps, Temporal logic for scenario-based specifications, in: Proc. 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '05), in: LNCS, vol. 3440, 2005, pp. 445–460.
[13] J.L. Peterson, Petri Nets, ACM Comput. Surv. 9 (3) (1977) 223–252.
[14] W. Reisig, Petri Nets: An Introduction, Springer-Verlag New York, Inc, New York, NY, USA, 1985.
[15] W. Reisig, G. Rozenberg, Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, in: LNCS, vol. 1491, Springer, 1998.
[16] W. Thomas, Automata on infinite objects, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), 1990, pp. 133–192.