Scenario-Based Analysis and Synthesis of Real-Time Systems Using Uppaal

Kim G. Larsen, Shuhao Li, Brian Nielsen, Saulius Pusinskas
Center for Embedded Software Systems (CISS)
Aalborg University
DK-9220 Aalborg, Denmark
Email: {kgl, li, bnielsen, saulius}@cs.aau.dk

Abstract—We propose an automated, tool-supported approach to scenario-based analysis and synthesis of real-time embedded systems. The inter-object behaviors of a system are modeled as a set of live sequence charts (LSCs), and the scenario-based user requirement is specified as a separate LSC. By translating the set of LSC charts into a behavior-equivalent network of timed automata (TA), we reduce the problems of model consistency checking and property verification to classical CTL real-time model checking problems, and reduce the problem of centralized synthesis for open systems to a timed game solving problem. We implement a prototype LSC-to-TA translator, which can be linked to existing real-time model checker UPPAAL and timed game solver UPPAAL-TIGA. Preliminary experiments on a number of examples show that it is a viable approach.

I. INTRODUCTION

In the early stage of model-based development of real-time embedded systems, it is desirable to prototype a system and its operating environment using a number of use cases and scenarios. A next round refinement of the system model and the transition from model to implementation can start only if the model is checked to be consistent (i.e., implementable), and correct with respect to some specified scenario-based user requirements. Furthermore, from a correct-by-construction perspective, as a part of the long-known dream for “automated system design”, we may wish to synthesize executable object systems from scenario-based descriptions.

Live Sequence Chart (LSC) [1] is a scenario-based modeling (i.e., as driving chart) and requirement specification (i.e., as monitored chart) language, which can describe systems in an assume-guarantee style. A universal LSC chart can optionally contain a prechart that specifies the scenario which, if successfully executed, forces the system to satisfy the scenario given in the actual chart body (the main chart). LSC extends Message Sequence Chart (MSC) mainly by adding modalities 1. The rich language facilities and the unambiguous semantics make LSC a nice visual formalism for early-stage characterization of distributed, real-time, and embedded systems.

Scenario-based approaches that use LSCs enjoy the advantage of piecewise incremental construction of system models.

However, model consistency checking and property verification (i.e., to check whether an LSC-modeled system satisfies a scenario-based requirement) are difficult due to the need to consider both the explicitly as well as the implicitly specified behaviors in each scenario, and the interplays among the different scenarios. Scenario-based synthesis is difficult because all possible scenario interactions have to be considered. The problems become even more complicated for real-time systems, as time-enriched LSCs may contain subtle timing errors that are difficult to diagnose.

Powerful verification techniques and tools are utilized to assist scenario-based analysis and synthesis for LSCs, e.g. in smart play-out [2], [3], [4], satisfiability checking [5], [4], consistency checking [5], and synthesis [6], [7], [8], [9], [10].

Some of these work address the problems mainly from a theoretical viewpoint [6], [9]. Some of them handle property verification with only existential charts [5], [4]. A number of them have no real-time support [6], [2], [5], [8], [9]. While synthesis for LSC-modeled real-time systems is supported in [7], it is incomplete in the sense that some systems are announced not synthesizable while they actually are. The reason is that the smart play-out mechanism [2], [3] which [7] relies on implements only a local consistency checking where the Play-Engine looks only one super-step ahead in the LSC state space. A recent work [10] tackles this by employing controller synthesis techniques to achieve complete consistency checking. However as an extension to smart play-out, this method, like the earlier work [7], is limited to discrete time and a restricted form of timing constraints 2.

In this paper we equip a subset of the LSC language with timed automaton (TA)[11]-like real-valued clock variables and clock constraints, and define a trace-based semantics. We use a set of driving universal charts (collectively called an LSC system) to model the inter-object (or inter-process) interaction behaviors of the system in question, and use a monitored universal/existential chart to specify the user requirement. We translate the LSC system into a behavior-equivalent network of interacting timed automata, which can properly mimic the important LSC features such as liveness and intra/inter-

1The existential and cold (resp. universal and hot) modalities represent the provisional (resp. mandatory) global and local requirements, respectively. An existential chart eLSC (resp. universal chart uLSC) specifies restrictions over at least one satisfying (resp. all possible) system runs. A cold condition may be violated, whereas a hot one must be satisfied.

2In the original definition of time-enriched LSC [1], the special Clock object has a property Time that is an integer variable, and a method Tick that each time increases Time by 1. Timing constraints take the form of only "Time op (time variable + delay expression)", where op is a relational operator.
chart coordinations. The problems of consistency checking and property verification are reduced to real-time model checking problems in UPPAAL [12]. Furthermore, by viewing the interaction between the controllable system processes ($Ssys$) of an LSC system and their “hostile” (uncontrollable) environment processes ($Env$) as playing a game, we reduce the problem of scenario-based synthesis for open systems (i.e., $Sys$) to a game solving problem. The timed game solver UPPAAL-TIGA [13] can be employed to check the solvability, and if yes, to generate a strategy for $Sys$. Compared with existing work, our approach of analysis and synthesis features:

- TA-like clock variables and timing constraints (compared with [3], [7], [4], [10]);
- Complete consistency checking and synthesis (comp. [6], [7]);
- Property verification with uLSCs (comp. [5], [4]); and
- Automated, tool-supported approach (comp. [6], [9]).

II. LIVE SEQUENCE CHARTS

In this paper, LSC in its simplest form is a message-only untimed chart (Fig. 1), i.e., there are only elements of instance lines, locations, messages, and precharts/main charts. In our time-enriched version (Fig. 2), there are further elements of (clock) variables, conditions (clock constraints), assignments (clock resets).

Specifically, along each $I_i$ there are four standard positions $StdPos(L, I_i) = \{Pch_top, Pch_bot, Mch_top, Mch_bot\} \subset pos(L, I_i)$, denoting the entry/exit points of the prechart/main chart, respectively, such that: (1) $0 = Pch_top < Pch_bot < Mch_top < Mch_bot = p_{max_{L, I_i}}$; and (2) $Pch_bot + 1 = Mch_top$.

A location is a position on a certain instance line. The set of all locations of chart $L$ are denoted as $Loc = loc(L) = \{(I_i, p) \mid I_i \in inst(L), p \in pos(L, I_i)\}$. The set of message-anchoring locations of $L$ are denoted as: $loc_M(L) = \{(I_i, p) \mid I_i \in inst(L), p \in pos(L, I_i) \setminus StdPos(L, I_i)\}$.

Let $\Sigma = msg_L(L)$ be the message alphabet of $L$. A message occurrence (or “message” for short) $mo = ((I_i, p), m, (I_{i'}, p'))$ corresponds to instance $I_i$, while in its position $p$, sending signal $m \in \Sigma$ to instance $I_{i'}$ at its position $p'$. We call $lab(mo) = m$ the message label, $head(mo) = (I_i, p')$ and $tail(mo) = (I_{i'}, p)$ the message head and tail locations, and $src(mo) = I_i$ and $dest(mo) = I_{i'}$ the source and destination instances, respectively. We use $loc(mo) = \{head(mo), tail(mo)\}$ to denote the message anchoring locations. The set of all messages in chart $L$ are denoted as $M = msg_L(L) \subseteq \{(I_i, p), m, (I_{i'}, p')\} \in loc_M(L) \times \Sigma \times loc_M(L) \mid i \neq i', p \leq StdPos(L, I_i).Pch_bot \Leftrightarrow p' \leq StdPos(L, I_{i'}).Pch_bot\}.

In our time-enriched LSCs (Fig. 2), each $m$-labeled message can be optionally associated with a condition (i.e., a conjunction of clock constraints) $g$, and/or an assignment (i.e., a set of clock resets) $a$. The intuitive meaning of message synchronization $[g]m/a$ from location $(I_i, p)$ to $(I_{i'}, p')$ is that, if immediately before this synchronization the clock valuation $v$ satisfies $g$, then this synchronization can fire; and immediately after the firing, $v$ will be updated according to $a$ (we view $a$ as a transformer, and denote the new valuation as $v' = a(v)$). A condition $g$ has a temperature, denoted $g.temp$, which can be cold or hot in a main chart, and only cold in a prechart. We assume that any condition and assignment is associated with a certain message.

In chart $L$, each location is either associated with a message, or it is an entry/exit point of the prechart/main chart. We define a labeling function $\lambda : Loc \rightarrow M \cup \{nil\}$ to map a location of the former type to its corresponding message, and a location of the latter type to $nil$.

Locations in a chart are partially ordered as follows:

- Along each $I_i$: location $l$ is above $l' \Rightarrow l \leq l'$;
- Given a message $mo$, if $l = head(mo)$ and $l' = tail(mo)$, then $l$ and $l'$ have the same order: $\forall mo \in M, \forall l, l' \in Loc. (\lambda(l) = mo) \land (\lambda(l') = mo) \Rightarrow (l \leq l') \land (l' \leq l)$.

The partial order relation $\leq \subseteq Loc \times Loc$ is defined as a transitive closure of $\leq$.

**Definition 1.** A cut of a chart $L$ is a downward-closed set of locations that span across all the instance lines in $L$. Downward-closure means that if a location $l$ is included in the cut $c \subseteq Loc$, so are all of its predecessor locations: $\forall c \subseteq Loc, \forall l, l' \in Loc. (l \in c \land l' \leq l) \Rightarrow l' \in c$. 

![Chart 1](image1.png)

![Chart 2](image2.png)

Fig. 1. Two consistent untimed charts.

![Chart 1](image3.png)

![Chart 2](image4.png)

Fig. 2. Two consistent time-enriched charts.

In this paper we assume the synchrony hypothesis, i.e., system events consume no real time, and time may elapse only between events. We consider the invariant mode of chart activation, i.e., the chart will be activated whenever the prechart is matched, regardless of the state of the system.

A universal LSC (uLSC) has a main chart ($Mch$). In this paper we assume that it also has a prechart ($Pch$).

Given a uLSC $L$, let $I = inst(L)$ be the set of instance lines (i.e., processes) in $L$. Along each instance line $I_i \in I$ there are a finite set of “positions” $pos(L, I_i) = \{0, 1, 2, \ldots, p_{max_{L, I_i}}\} \subset \mathbb{N}$. See Fig. 1(a), black bubbles.
Given a cut $c \subseteq \text{Loc}$ and a message $mo \in M$, we say $mo$ is enabled at cut $c$, denoted $c \xrightarrow{mo}$, if, $\forall l' \in c$, $l'' \in \text{loc}(mo). (l \notin c) \land (c' = c \cup \text{loc}(mo)). (l' \leq l \land l'' = l')$.

A cut $c'$ is an $mo$-successor of cut $c$ if, $mo$ is enabled at $c$, and $c'$ is achieved by adding the set of locations that $mo$ anchors into $c$, or formally, $(c \xrightarrow{mo}) \land \forall l \in \text{loc}(mo). (l \notin c) \land (c' = c \cup \text{loc}(mo))$.

A cut $c$ is minimal, denoted $\top$, if each location in $c$ is a top location of a certain instance line; $c$ is maximal, denoted $\bot$, if the bottom locations of all instance lines are included in $c$. Specifically, the minimal and maximal cuts of the prechart (resp. main chart) are denoted $Pch.\top$ and $Pch.\bot$ (resp. $Mch.\top$ and $Mch.\bot$), respectively. A $\top$ or $\bot$ cut represents a compulsory synchronization for all instance lines. Thus $\preceq$ on Loc is extended as follows:

- All locations in the frontier of the same minimal or maximal cut have the same order, $\forall c \in \{Pch.\top, Pch.\bot, Mch.\top, Mch.\bot\}. \forall l', l'' \in c. (l \leq l') \land (l'' \leq l)$.

A cut $c$ together with a valuation $v$ of all the clocks is called a configuration $(c, v)$ of a time-enriched LSC chart. A universal chart starts from the initial configuration, advances from one to the next configuration, until a hot violation occurs, or until the chart arrives at the maximal cut configuration, and then starts all over again (i.e., to begin a next round execution).

There could be three types of advancement steps between two configurations $(c, v)$ and $(c', v')$ of an LSC:

- **Message synchronization step.** Given an $m$-labeled message $mo$ which is optionally associated with a condition $g$ and/or an assignment $a$, there is a message synchronization step $(c, v) \xrightarrow{mo} (c', v')$ if,
  - (normal advancement), $v \models g$, $c'$ is an $mo$-successor of $c$, and $v' = a(v)$; or
  - (cold advancement), $c' = Pch.\top$, $v' = v$, and (either $mo$ violates $\prec$ in Pch, or $v \not\models g \land g.\text{temp} = \text{cold}$);

- **Silent step.** There is a silent step $(c, v) \xrightarrow{\_} (c', v')$ if, $v' = v$, and either
  - $(c = Pch.\bot$ and $c' = Mch.\top$); or
  - $(c = Mch.\bot$ and $c' = Pch.\top$); or
  - $c'$ is reached because some instance line moves from its top location to its bottom location in Pch or Mch autonomously (in case an instance line does not interact with others in Pch or Mch);

- **Time delay step.** There is a time delay step $(c, v) \xrightarrow{d} (c', v')$ where $d \in R_{\geq 0}$ if, $c' = c$, $v' = v + d$, and whenever there are messages that are enabled at cut $c$, then there exists at least one such message $mo$ such that its guarding condition $mo.g$ (if any) will still be satisfied after delay $d$, i.e., $\exists mo \in M.(v + d) \models mo.g$.

Similarly, if in the main chart, $mo$ violates $\prec$, or $v \not\models g \land g.\text{temp} = \text{hot}$, then it is a hot violation, denoted $(c, v) \xrightarrow{\_}$.

**Definition 2.** A run of a time-enriched universal LSC chart is a sequence of configurations $(c^0, v^0) \cdot (c^1, v^1) \cdot \ldots$ that are connected by the advancement steps, i.e., $\forall i \geq 0, \exists u_i \in (\Sigma \cup \{\tau\} \cup R_{\geq 0}). (c^i, v^i) \xrightarrow{u_i} (c^{i+1}, v^{i+1})$.

The transition relation $\rightarrow$ as mentioned above each time consumes only a single letter $u \in (\Sigma \cup \{\tau\} \cup R_{\geq 0})$. We extend it to $\rightarrow^*$ such that it consumes a (finite or infinite) word $w \in (\Sigma \cup \{\tau\} \cup R_{\geq 0})^*$.

For an entire LSC system $LS = \{L_i \mid 1 \leq i \leq n\}$, its message alphabet is $\Pi = \bigcup_{i=1}^{n} \Sigma_i = \bigcup_{i=1}^{n} msg_A(L_i)$.

**Definition 3.** A timed trace $\gamma \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^* \cup (\Pi \cup \{\tau\} \cup R_{\geq 0})^\omega$ satisfies an LSC prechart or main chart $C$ if its projection $\gamma|_{(\Sigma \cup \{\tau\} \cup R_{\geq 0})}$ has a prefix $\mu$ which is the accepted word of a run that successfully exercises $C$, and no prefix of it leads to a hot violation, i.e., $\gamma \models C \Rightarrow (\exists u \in (\Sigma \cup \{\tau\} \cup R_{\geq 0})^* \land \exists m \in (\Sigma \cup \{\tau\} \cup R_{\geq 0})^\omega. (\gamma|_{(\Sigma \cup \{\tau\} \cup R_{\geq 0})} = \mu \cdot m \cdot \xi \land (\top, v^0) \xrightarrow{\_} (\bot, v^\omega)) \land (\exists \mu'. (\gamma|_{(\Sigma \cup \{\tau\} \cup R_{\geq 0})} = \mu \cdot m \cdot \xi \land (\top, v^0) \xrightarrow{\_} \mu') \land (\bot, v^\omega)$.

A finite trace $\gamma \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^\omega$ satisfies chart $C$ exactly, denoted $\gamma \models C$, iff $(\gamma \models C) \land (\forall \mu'. (\gamma|_{(\Sigma \cup \{\tau\} \cup R_{\geq 0})} = \mu) \land (\top, v^0) \xrightarrow{\_} \mu')$.

**Definition 4.** A timed trace $\gamma \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^\omega$ satisfies (passes) a universal chart $L$ iff, whenever a finite sub-trace matches the prechart, then the main chart is matched immediately afterwards, $\gamma \models L \Rightarrow \forall \alpha, \mu \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^* \land \exists \beta \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^\omega (\alpha \cdot \mu \cdot \beta = \gamma) \land (\mu \models Pch) \Rightarrow \beta \models Mch$.

In a run of an LSC system, a message synchronization step corresponds to the firings of all identically labeled enabled messages, (at most) one in each chart.

**Definition 5.** A timed trace $\gamma \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^\omega$ satisfies (passes) an LSC system $LS$ iff, $\gamma$ is an infinite run of $LS$, and it satisfies each chart $L_i$ in $LS$ separately.

III. LSC to TA TRANSLATION

A. Mapping LSC instance to Uppeaal TA

An LSC chart $L$ characterizes a scenario of object interactions in a communication system. The set of instances (objects) in $L$ can be viewed as a set of parallelly executing processes that collaborate to achieve a common goal as specified by $L$. Since Uppeaal operates on a network of parallelly composed processes (TA) that communicate with each other, this motivates us to translate each instance line of $L$ into a timed automaton. This idea in spirits resembles [2], [5].

Basic structure mapping

Each instance line $I_i$ in chart $L_i$ of the LSC system $LS$ is mapped into a timed automaton $A_{u,i}$. Each position on $I_i$ corresponds to a TA location in $A_{u,i}$, and each discrete
advancement step (i.e., a message synchronization step, or a silent step) on \(I_i\) corresponds to a TA edge in \(A_{ui,i}\). The sending (resp. receiving) of an \(m\)-labeled message corresponds to an \(m!\) (resp. \(m?)\)-labeled TA edge in \(A_{ui,i}\).

**Enforcing liveness of main chart**

Intuitively, once an instance line progresses into its portion in \(Mch\), then the message sending (if any) along this instance line must happen. Technically, if at a position on the main chart portion of \(I_i\) there is a message sending, then we mark the corresponding message emitting TA location as an **urgent** location. Since in an urgent location time is frozen and thus delay is not allowed, this will force the message to be emitted in a naive way.

**Handling intra/inter-chart coordination**

In the prechart (resp. main chart) of an LSC chart, once all the participating instance lines have progressed to their \(Pch\_bot\) (resp. \(Mch\_bot\)) positions, then the prechart (resp. main chart) is successfully matched. In this case, the prechart (resp. main chart) will be exited immediately, and the main chart (resp. prechart) will be entered immediately afterwards. To synchronize all the participating instance lines for such a prechart/main chart (resp. main chart/prechart) transfer, for each LSC chart, we create a dedicated (auxiliary) coordinator automaton. This automaton will communicate with the automata for the instance lines by using auxiliary binary synchronization channels such that it can bookkeep how many instance lines are done with their prechart (resp. main chart) portions. Once the coordinator automaton realizes that the prechart (resp. main chart) has been successfully matched, it will launch a broadcast synchronization with the automata of all the relevant instance lines immediately.

In scenario-based modeling, an event can be described by multiple scenarios. To be more specific, it is possible that more than one LSC chart can have \(m\)-labeled messages from instance \(A\) to instance \(B\). If these messages are all enabled and one of them is chosen to be fired, then all the others must also be fired at the same time. Clearly, this requires a kind of inter-chart coordination. To achieve this, we use broadcast synchronization channels rather than binary synchronization channels for these messages. In the translated timed automata, if there is an \(m!\)-labeled edge from one to another TA location, then we add an \(m?\)-labeled edge between these two TA locations to “accompany” the \(m!\)-labeled edge.

**Handling cold and hot violation**

Once a cold violation occurs, we let the message sender report to the coordinator automaton in charge, which in turn immediately synchronizes the automata for all the relevant instance lines for a “reset”. Once a hot violation occur, we set the global flag boolean variable **hotviolated** to true.

**Dealing with time**

To mimic the behaviors of a clock constraint and clock reset in an LSC chart, we use a linked sequence of TA edges, whose atomicity is ensured by the **UPPAAL** feature of committed location. Moreover, the upper bounds of clock constraints are extracted to be used as TA location invariants, which will “force” the relevant messages to be emitted within due time frames.

**Translating environment processes**

The processes (instance lines) in an LSC system can be partitioned into two sets: the environment processes (\(Env\)), and the system processes (\(Sys\)). Messages sent from \(Env\) to \(Sys\) processes are uncontrollable, whereas other message sendings are controllable. To model this, we mark the message-sending edges in the translated timed automata of the \(Env\) processes as uncontrollable edges (in dashed lines), and other edges as controllable edges (in solid lines).

**Translating monitored charts**

In comparison with a driving universal chart, the translation of a monitored chart into timed automata is different in the point that a monitored chart only “listens to” the messages of a monitored chart into timed automata is different in the translation of \(LSC\) system, and never emits messages itself. When translating such a chart into a network of timed automata, if at position \(s\) of instance line \(I_k\) there is a sending of an \(m!\)-labeled message, then we add an \(m?\)-labeled TA edge from \(l_{s-1}\) to \(l_s\), and not an \(m!\)-labeled one.

The detailed translation rules and the translation examples can be found in a longer version of this paper [14].

**B. Complexity of translated TA**

Let \(\mathcal{LS}\) be a set of time-enriched LSC charts \(L_1, L_2, \ldots, L_n\), and let \(NTA_{\mathcal{LS}}\) be the translated network of timed automata. Let \(\text{Inst}(L_i), \text{msg}_A(L_i)\) and \(\text{msg}(L_i)\) denote the set of instance lines, the message alphabet, and the set of message occurrences of chart \(L_i\), respectively. After translation, we have the following numbers of:

- **TAs:** \(\sum_{i=1}^{n} |\text{Inst}(L_i)| + 1 + |\bigcup_{i=1}^{n} \text{msg}_A(L_i)|\);
- **Channels:** \(\sum_{i=1}^{n} (2 \cdot |\text{Inst}(L_i)| + 4) + 4 \cdot |\bigcup_{i=1}^{n} \text{msg}(L_i)|\); (worst-case only)
- **Auxiliary variables:** \(4 \cdot |\bigcup_{i=1}^{n} \text{msg}_A(L_i)| + 2n + 1 + 2 \cdot \sum_{i=1}^{n} |\text{msg}(L_i)|\).

Analysis of the complexity of the translated network of timed automata can be found in [14].

**C. Behavior equivalence of translation**

Let \(\Pi\) be the message alphabet of \(\mathcal{LS}\), and \(\text{Act} = \Pi \cup \text{Aux}\) be the normal and auxiliary channels in \(NTA_{\mathcal{LS}}\).

**Theorem 1.** \(\forall \gamma_1 \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^*.(\gamma_1 \models \mathcal{LS}) \Rightarrow \exists \gamma_2 \in (\text{Act} \cup \{\tau\} \cup R_{\geq 0})^*.((\gamma_2 \models \mathcal{NTA}_{\mathcal{LS}}) \wedge (\gamma_2|_{\Pi \cup \{\tau\} \cup R_{\geq 0}} = \gamma_1|_{\Pi \cup \{\tau\} \cup R_{\geq 0}}))\), and \(\forall \gamma_1 \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^*.((\gamma_1 \models \mathcal{NTA}_{\mathcal{LS}}) \Rightarrow \exists ! \gamma_2 \in (\Pi \cup \{\tau\} \cup R_{\geq 0})^*.((\gamma_2 \models \mathcal{LS}) \wedge (\gamma_2|_{\Pi \cup \{\tau\} \cup R_{\geq 0}} = \gamma_1|_{\Pi \cup \{\tau\} \cup R_{\geq 0}}))).\)

**Theorem 1** indicates that each accepted timed trace \(ttr\) in \(\mathcal{LS}\) uniquely corresponds to a cluster of accepted timed traces

\(^4\)A committed TA location is an urgent location whose outgoing transitions have higher priority to be taken than those from non-committed ones.
in $NTA_{LS}$. All these traces and $trt$ project to exactly the same trace on $(I(II \cup R_{\geq 0})$.

The semantics of timed automata, and the proofs of the theorems in this paper can be found in [14].

IV. SCENARIO-BASED ANALYSIS AND SYNTHESIS

A. Consistency checking

An LSC system is inconsistent iff the system model has internal contradictions [6], i.e., there does not exist an infinite message sequence that satisfies all uLSCs. Alternatively, an LSC system is inconsistent iff, along all paths, whenever the main chart of a certain LSC chart is activated, then along all possible paths there will eventually be a hot violation of the main chart of some particular LSC chart (i.e., the flag boolean variable hotviolated will be set true).

UPPAAL uses a fragment of the CTL logic as its query language. Formulas could take the forms $E\diamond \phi, E\Box \phi, A\diamond \phi, A\Box \phi$, where $\phi, \phi_1$, and $\phi_2$ are state formulas. In particular $\phi \rightarrow \phi_2$ is a shorthand for $A\Box (\phi \Rightarrow A\diamond \phi_2)$, which characterizes the assume-guarantee style liveness.

**Theorem 2.** $LS = \{L_1, L_2, \ldots, L_n\}$ are inconsistent $\iff$ $NTA_{LS} \models (\forall i=1) Coord_i$.$Mch_{top} \rightarrow$ (hotviolated = true).

In the above theorem, $Coord_i$.Mch_top means that the coordinator $TA$ Coord_i for chart $L_i$ is in location Mch_top, which indicates that the main chart of $L_i$ is activated.

For example, Fig. 1 are two consistent universal LSC charts, whereas Fig. 3 are two inconsistent ones.

![Fig. 3. Two inconsistent universal charts.](image)

**B. Property verification**

Property verification asks whether a system that is modeled as a set of driving uLSCs $LS$ satisfies the requirements that are specified in a monitored universal or existential LSC chart $L'$. Here $L'$ will be translated into a network of observer timed automata $NTA_{L'}$.

**Theorem 3.** Let $L'$ be a monitored universal chart. $LS \models L' \iff (NTA_{LS} || NTA_{L'}) \models Coord_{L'}.Mch\_top \rightarrow Coord_{L'}.Mch\_bot$.

Theorem 3 says that whenever the prechart of the monitored universal chart $L'$ is matched, the main chart of $L'$ must be matched afterwards. This reflects the liveness nature of the requirements that are expressed by monitored universal charts.

As an example, the (single chart) LSC system in Fig. 4(a) satisfies the requirement specified in Fig. 4(b).

![Fig. 4. $L_1$ (model) satisfies $L_2$ (property).](image)

**Theorem 4.** Let $L'$ be a monitored existential chart. $LS \models L' \iff (NTA_{LS} || NTA_{L'}) \models E\Box Coord_{L'}.Mch\_bot$.

Theorem 4 says that in order to satisfy the monitored existential chart $L'$, there should exist at least one trace that successfully exercises $L'$. This reflects the existence nature of the requirements that are expressed by monitored existential charts.

**C. Synthesis for open systems**

A timed automaton with its edges partitioned into controllable and uncontrollable ones is called a *timed game automaton* (TGA) [15]. A network of parallelly composed timed game automata for $E_{NV}$ and $S_{YS}$ can be viewed as a timed game structure: the timed game automata for $S_{SYS}$ as a player masters the set $A_{c}$ of controllable edges, and the timed game automata for $E_{NV}$ (denoted $NTA_{E_{NV}}$) as the opponent masters the set $A_{nv}$ of uncontrollable edges. Given a winning objective, $NTA_{SYS}$ will take moves in order to win the game (i.e., to bring the system into a winning state), whereas $NTA_{E_{NV}}$ may spoil the game.

Let $S$ be the state space of $NTA_{E_{NV}} || NTA_{SYS}$, and $\epsilon \notin (A_{c} \cup A_{nv} \cup \{\tau\})$ be an empty action which means ‘do nothing at this moment’. A state-based (or memoryless) strategy for $NTA_{SYS}$ is a (partial) function $\rho : S \rightarrow (A_{c} \cup \{\epsilon\})$, which constantly guides $NTA_{SYS}$ to take appropriate controllable actions, or just delay (and wait for the semantic state to be changed by an uncontrollable action of $NTA_{E_{NV}}$, or by the elapse of time).

UPPAAL-TIGA [13] is a timed game solver. Its inputs include a set of timed game automata, and a winning objective formulated as an extended ACTL (the universal fragment of CTL) formula. For example, property “control: $A\Box \phi$ asks whether there exists a strategy $\rho$ for $NTA_{SYS}$ such that if $NTA_{SYS}$ is supervised by $\rho$, then $NTA_{LS}$ is guaranteed to always (i.e. invariably) satisfy $\phi$, no matter how $NTA_{E_{NV}}$ behave. If the property is satisfied, UPPAAL-TIGA will be able to synthesize a winning strategy for $NTA_{SYS}$.

**Theorem 5.** An executable object system for $S_{SYS}$ can be synthesized $\iff$ $NTA_{LS} \models A\Box$ (hotviolated = false).

Synthesis for $S_{SYS}$ is possible only if the entire system $LS$ can be guaranteed not to be hot-violated no matter how the $E_{NV}$ processes behave. By means of behavior equivalence, this boils down to finding a winning strategy $\rho$ for $NTA_{SYS}$. Since the strategy (if ever exists) will oversee all the $S_{SYS}$ processes rather than being distributed to supervise each of them individually, it is a kind of centralized synthesis. It is clear that $NTA_{LS}$ as supervised by $\rho$ constitute one such desired executable object system.
V. TOOL IMPLEMENTATION

Fig. 5 shows our scenario-based analysis and synthesis framework. Among those inputs (the shadowed elements), the \( \text{Env/Sys} \) partitioning directives specify which processes in the charts of \( \mathcal{LS} \) belong to \( \mathcal{Env} \) and \( \mathcal{Sys} \), respectively.

We build a GUI-based LSC editor, with which we can construct either driving LSC charts or monitored LSC charts. A prototype LSC-to-TA translator has been implemented, which is capable of batch translation of driving and monitored charts, and translation for the environment processes. The translator-generated timed automata and CTL formulas comply with the UPPAAL timed automaton and query language syntaxes, and can thus be fed into UPPAAL and UPPAAL-TIGA directly.

Preliminary experiments have been conducted on an Intel- ligent Mouse (\# of charts, instances, message labels: 5, 3, 5) and an ATM Machine (20, 3, 6) examples, and a DHCP (Dynamic Host Configuration Protocol) case study (71, 3, 16). Compared with the verification and game solving of the translated network of TAs, the translation process itself has negligible time overheads and memory consumptions. This is reasonable. The complexity of scenario-based analysis and synthesis mainly lies in the LSC models themselves. As a syntactical level manipulation, our translation only introduces some auxiliary channels and bookkeeping variables (not clock variables) to implement the LSC semantics.

VI. CONCLUSIONS

We present a timed extension to a subset of the LSC language, and define a trace-based semantics. We transform LSCs into a network of behavior-equivalent timed automata. The LSC consistency checking, property verification and synthesis problems can be reduced to CTL real-time model checking and timed game solving problems. We implemented an LSC editor and a prototype LSC-to-TA translator. When linked with existing real-time model checker UPPAAL and timed game solver UPPAAL-TIGA, they constitute a tool chain for automated, scenario-based analysis and synthesis of real-time systems. Preliminary experiments on a number of examples show that it is a viable approach.

The complexity of scenario-based approaches mainly lies in the interplays of a large number of simple charts. Our translations avoid constructing a global state machine. Rather the intricate semantics of LSC chart progress and intra/inter-chart coordinations are mostly left up to UPPAAL.

As future work, we may consider more LSC constructs, such as co-region, symbolic instance, control structures, etc. The implementation of a full-fledged translator, and the application of the tool chain to industrial case studies are desirable. Furthermore, scenario-based synthesis for systems with imperfect information (e.g., some uncontrollable actions are not observable) is also worth investigation.

REFERENCES