Visual Formal Specification using (N)TLCharts: Statechart Automata with Temporal Logic and Natural Language Conditioned Transitions

Doron Drusinsky
Time-Rover, Inc., 11425 Charsan Ln., Cupertino, CA 95014, USA
www.time-rover.com

Abstract

This paper describes TLCharts, a visual specification language that combines the visual and intuitive appeal of non-deterministic Harel Statecharts with formal specifications written in Linear-time (Metric) Temporal Logic (LTL and MTL). The formalism is described using a practical infusion pump requirement example. The infusion pump TLChart specification is then compared with two competing representations: temporal logic and deterministic Harel statecharts. The infusion pump example will also be used to point out the strength of each constituent TLCharts component. We provide an informal semantics for TLCharts using non-deterministic automata with negation and overlapping states. Finally, we show how natural language snippets are used instead of TLChart temporal logic conditions thereby inducing a formalism we call NTLCharts.

1 Introduction

Temporal Logic is a special branch of modal logic that investigates the notion of time and order. Linear-time Temporal Logic (LTL) is an extension of propositional logic where, in addition to the well-known propositional logic operators, there are four future-time operators (◊-*Eventually*, □-*Always*, U-*Until*, O-*Next*) and four dual-past time operators. Pnueli [Pn] suggested using LTL for reasoning about concurrent programs. Since then, several researchers have used LTL to state and measure correctness of concurrent programs, protocols, and hardware (e.g., [MP, Pn]). Metric Temporal Logic (MTL) was suggested by Chang, Pnueli, and Manna as a vehicle for the verification of real time systems [CPM]. MTL extends LTL by supporting the specification of relative-time and real-time constraints. With MTL, all four LTL future-time operators can be characterized by relative-time and real-time constraints specifying the duration of the temporal operator. Temporal Logic with Time Series constraints (TLS) was suggested by Drusinsky as an extension of MTL which enables temporal specifications that assert about time-series properties such as stability, monotonicity, and min-max values [D2].

Ever since first published [Ha] and later incorporated into the OMT methodology and eventually into the UML standard, Harel statecharts have been described in numerous papers and books (e.g.[Br, RB]). Statecharts extend finite state diagram with hierarchy (state nesting), concurrence, and history states. Harel Statecharts are typically used for design analysis and implementation; for example, Brugge suggests using statecharts in the design analysis phase of an object oriented UML based design methodology [Br]. Theoretical results [DH] show that non-deterministic are exponentially more succinct than deterministic Harel statecharts.

A formal semantics of Harel statecharts has been suggested in [HN]. This paper uses new automata theoretic semantics for statecharts first suggested in [D1]. This semantics lends itself to the inherently non-deterministic semantics required by the TLCharts formalism.

Sowmya and Ramesh suggested in [SR] to use temporal logic assertions with statechart qualities by applying temporal logic in a hierarchical manner; the resulting language is a new hierarchical form of the textual temporal logic formalism. In comparison, our hybrid language is a true automata-theoretic hybrid with a unified syntax and semantics; the resulting language is highly visual and familiar language, with special LTL annotation of some transitions.

Enciso et-al. [E] suggest LNint-e, a logic that combines points and intervals and the absolute and relative approaches of LTL and statecharts. This is a new logic with new syntax and semantics. In contrast, our suggested language maintains, for the most part, the syntax and semantics of both languages.

The Mathworks' Stateflow statechart tool has a so-called temporal logic extension. Stateflow events and conditions can use the four operators *after*, *before*, *at*, and *every*. These four operators are essentially extended versions of the LTL eventuality operator. Most notably, the Stateflow formalism lacks non-

Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)

IEEE Computer Society

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

determinism, negation, and an operator equivalent to LTL's *Until* operator.

Non-deterministic Finite Automate (NFA) are often used as a specification language [HU]. The TLChart formalism suggested in this paper extends the NFA formalism in two ways: it suggests using non-deterministic statecharts with negation instead of flat and sequential NFA formalism, and it supports the annotation of transitions with LTL, MTL and TLS conditions.

In this paper we describe TLCharts, a formalism that visually and intuitively resembles Harel statecharts while enabling temporal-logic conditioned transitions. This is useful for the specifying abstract non-deterministic temporal properties inside a statechart specification. This paper contains informal semantics of TLCharts; formal semantics are avaliable in [D3].

2 Example: Infusion Pump Keypad Control

The infusion pump example consists of the following four conditions: (infusion)*begin*, (infusion)*end*, *keyPressed* and *alarm*. The requirement is:

R1: a session is the interval between a *begin* and an *end-condition.* For every such session a *keyPressed* must be repeatedly sensed within two-minute intervals or else an *alarm* must sound within 10 seconds and until *keyPressed* is sensed. Also according to this specification, once the alarm sounds then the assertion has succeeded and no more alarms are permitted. The *end-condition* is defined as an *end* being repeatedly sensed until a later time when *begin* is sensed.

In sections 2.1 and 2.2 we analyze two seemingly correct formal specifications for requirement R1, namely MTL and Harel statechart specifications. These specifications are more complex than the alternate TLChart specification and also contain subtle inaccuracies rendering them less effective than the corresponding TLChart specification.

2.1 Infusion Pump: MTL Specification

The following MTL assertion attempts to capture requirement R1:

L1: $\Box$ (begin =>
L2: (((begin $\vee$ keyPressed) =>
L3: (($\Box_{\leq 120}$ $\neg$alarm) $\wedge$
L4: ((O $\Diamond_{\leq 120}$ keyPressed)
L5: $\vee$ ($\neg$keyPressed $U_{[120,130]}$
L6: (alarm U
L7: (keyPressed $\wedge$ $\Box\neg$alarm)))
L8:)
L9:)
L10:) U (end U
L11: (begin $\wedge$ end))
L12:))

Line L1 initiates the session. Line L2 combined with L4 guarantees the repetitive demand for *keyPressed* to be sensed every two minutes. Line L3 trivially requires no alarm until those two minutes have elapsed. Lines L5, L6, and L7 require an alarm within 10 seconds of those two minutes, and until *keyPressed* is sensed, with no alarm permitted afterwards. Lines L10 and L11 are for the end-condition .

This assertion suffers from several deficiencies:

1. The assertion is arguably non-trivial while the natural language requirement is straightforward. For example, the term *begin* $\vee$ *keyPressed* is confusing. In fact, for purposes of brevity, the MTL specification does not forbid an alarm while not in session; the Harel statechart of Fig. 1, and the TLChart of Fig. 2 do contain this constraint.
2. The assertion might fail under the following scenarios, assuming the LTL-model cycle time is one second, i.e., the assertion is evaluated every second. All scenarios begin with a *begin* at time 0.
   a. An interval of 122 seconds between two consecutive *keyPressed* events followed by an alarm sounding 1 second later and a *keyPressed* 1-second afterwards, followed by no *keyPressed* or alarm for 140 seconds. The assertion fails though the requirement is that following the first alarm the assertion must succeed.
   b. An interval of 122 seconds between two consecutive *keyPressed* events followed by an alarm sounding 4 seconds later and a *keyPressed* 1-second afterwards. The assertion fails because LTL's $\rho U \varphi$ requires ρ to repeatedly succeed until φ succeeds, namely $\neg$*keyPressed* must be constantly true until the alarm.
   c. An intuitive expectation is that an *end-condition* will terminate the need for a flow of *keyPressed* events. However, if *keyPressed* occurs at time *t* and an *end-condition* at time *t*+20 then one additional *keyPressed* will still be required after time *t*+20. In other words, there is no simple way to explicitly *truncate*

the requirement once an *end-condition* is detected other than to conjunct the *end-condition* with the inner parts of the rule. Separate research on LTL with truncated paths has been published in [EFHL].

2.2 Infusion Pump: Harel Statechart and TLChart Specifications

A deterministic Harel statechart specification of requirement R1 is illustrated in Fig. 1. and a corresponding TLChart specification is illustrated in Fig. 2. Section 3 describes the suggested informal syntax and semantics for TLCharts.

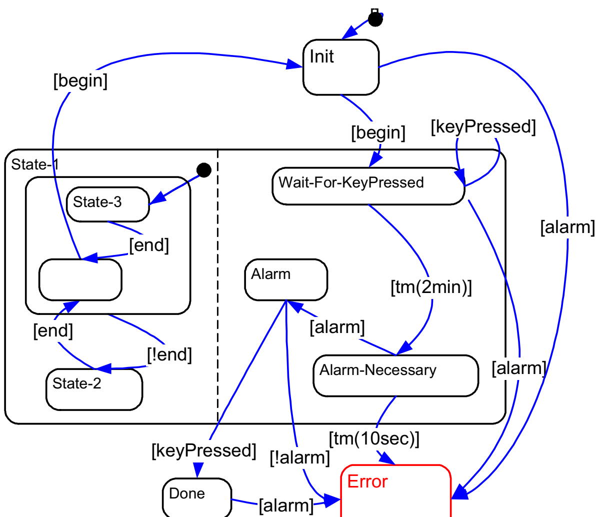

Figure 1. Deterministic Harel statechart specification for requirement R1.

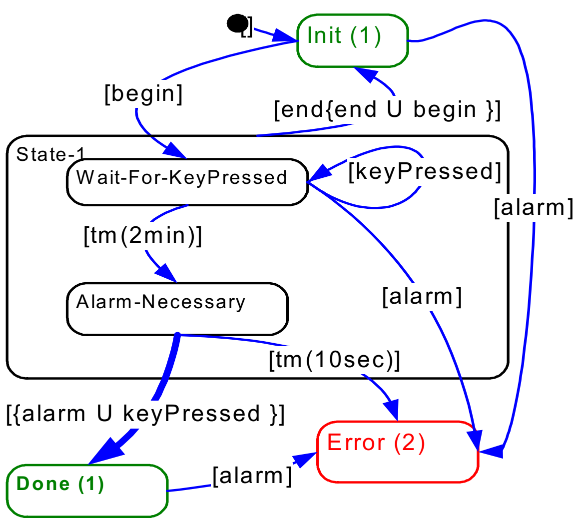

Figure 2. TLChart specification for requirement R1. All states other than *Error* are by default *good* states; all states with no specified priority have the default, i.e., lowest, priority.

Fig. 1 and Fig. 2 are both legal TLCharts, i.e. Harel statecharts are a special case of TLCharts, and so are LTL and MTL assertions. Note that TLCharts in Fig. 1 and Fig. 2 solve the problems described earlier.

A TLChart extends deterministic statechart in two primary ways:

1. Some transitions are annotated with LTL, MTL or TLS conditions, such as the transition labeled *alarm U keyPressed* in Fig. 2.

2. TLChart's support non-deterministic with negation. Armor plating of TLCharts, described in Section 5, uses this feature.

Note that Harel statecharts, when used for specification, must be deterministic; otherwise, the specification is ambiguous. Creating correct deterministic behavior is a non-trivial part of the implementation process. For example, consider the following scenario: *begin* at time 0 and then no *keyPressed* for more than two minutes followed, on cycle *#Cyc*, by the sequence: *Seq* = *end.alarm.keyPressed.begin*. The Harel statechart of Fig. 1 has unexpected behavior with respect to this scenario. Having *end* precede *alarm* indicates that the user wants to end the current session; nevertheless, the statechart ends the computation in state *Done* rather than in state *Init*. Consequently, a legal continuation of this scenario that results in a legal alarm will be determined by the statechart of as an error. A more accurate Harel statechart is a refinement of Fig. 1 with more implementation detail such that following *end*, whenever a sequence satisfying *alarm U keyPress* is recognized, it is memorized. Later, if the *end* turns out to be a false positive (i.e., the *end-condition* is not satisfied), the statechart will transition to state *Done*.

Alternatively, using statechart formalism with semantics that support non-determinism, the following non-deterministic approach can be used. When *end* is detected, then in addition to the existing computation leading towards state *Init*, a non-deterministic fork is made creating an additional computation that remains inside *State*-1; this computation kicks-in if the complete *end-condition* is not satisfied.

It will follow from the semantics of Section 3 that the TLChart of Fig. 2 operates on the input sequence *Seq* in the following accurate manner. The TLChart traverses the transition *State*-1→*Init* on cycle *#Cyc*, before the transition *Alarm-Necessary*→*Done* is enabled on cycle *#Cyc*+1.

It will also follow from the semantics that the TLChart of Fig. 2 is deterministic if *alarm* and *end*

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

are mutually exclusive; similar mutual exclusivity requirements exist for the Harel statechart of Fig. 1.

2.3 Infusion Pump Requirement R2

In preparation for the description of TLChart syntax and semantics we introduce an extension R2 to the R1 requirement as follows. Condition *valveOpen* (its negation denoted as *valveClosed*) is added as an additional visible condition. *The end-condition* is now re-defined as an interval that starts with the *valveClosed* and then *end* is repeatedly sensed until a later time when *begin* is sensed. The TLChart of Fig. 3 is an extension of Fig. 2 that formally captures requirement R2. It extends the TLChart of Fig. 2 with concurrence.

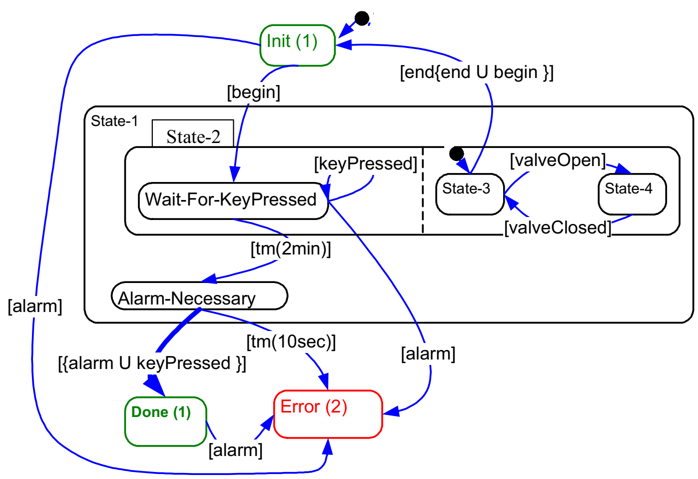

Figure 3. An extension of the TLChart of Fig. 2 that captures requirement R2.

3 TLCharts: Informal Syntax and Semantics

In this paper we consider Harel statecharts as first described in [Ha], including state hierarchy, concurrence, and history states. Hence, no state overlapping is permitted; this assumption will be changed in the next section. For simplicity, we assume that statechart transitions are annotated with conditions and not events, although we expect TLChart to be used and applied with events and conditions, much like UML statecharts. Hence, TLChart transitions are annotated with one or both of the following types of conditions: *propositional* and *temporal*. Temporal conditions include all legal LTL and MTL formulae. In Fig. 2, 3, and 4 temporal conditions are represented using curly braces. Hence [*end* {*end U begin*}] represents the propositional condition *end* and the temporal condition *end U begin*.

TLCharts specify requirements using formal languages. The semantics of a TLChart are defined using an *Equivalent Non-Deterministic Automaton* (ENFA) [D1, HU]. Once defined in terms of its ENFA, a TLChart defines correctness properties in a manner that resembles logic specification, such as temporal logic specification. It observes a given input tape and decides whether this tape is acceptable or not. In real life terms the input tape corresponds to a combined sequence of inputs to-, and manifested outputs from-, a given system.

The ENFA's state set consists of all possible state configurations in the original TLChart, i.e., where statechart concurrence is represented using all possible combinations *constituent* states from concurrent threads. Hence, in Fig. 3 {*Init*}, {*Wait-For-KeyPressed*, *State*-3}, and {Wait-For-*KeyPressed, State*-4}, are all legal states of the ENFA, while {*Init, State*-4} is not. Note that state configurations do not, in general, contain information about corresponding superstates, such as *Wait-For-KeyPressed* and *State*-3 residing under *State-2*, which in turn resides under *State-1*. This information is not necessary because, absent state overlapping, state hierarchy is unique. However, we will change this notation when we describe TLCharts with overlapping states.

As a preliminary step, before we describe the ENFA's transition relation, note that we can replace statechart and TLChart hierarchical transitions, such as *State*-1→*Init* in Fig. 2, with concurrence, using a new concurrent thread with one inner state, e.g. *State*-1*a*. The hierarchical transition is then replaced with the transition *State*-1*a*→*Init*.

To understand the ENFA's transition relation we first consider a TLChart with no temporal conditions. In this case the ENFA's transition relation pairs states (i.e., TLChart configurations) using one or more concurrent *constituent* TLChart transitions. Hence, in Fig. 3, several possible transitions are:

1. {*Wait-For-KeyPressed*, *State*-3}$\rightarrow_{\mathrm{keyPressed,\ valveOpen}}$ {*Wait-For-KeyPressed*, *State*-4} constructed from the concurrent firing of the non conflicting constituent TLChart transitions: *Wait-For-KeyPressed*$_{\rightarrow \mathrm{keyPressed}}$ *Wait-For-KeyPressed* and *State*-3$_{\rightarrow\ \mathrm{valveOpen}}$ *State*-4.
2. {*Wait-For-KeyPressed*, *State*-3}$\rightarrow_{\mathrm{keyPressed,}}${*Wait-For-KeyPressed*, *State*-3} constructed from the firing of the single constituent TLChart transition *Wait-For-KeyPressed*$\rightarrow_{\mathrm{keyPressed}}$ *Wait-For-KeyPressed.*

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

3. $\{$*Wait-For-KeyPressed*, *State*-3$\}\rightarrow_{\text{alarm}}$ $\{$*Error*$\}$ constructed from the single constituent TLChart transition *Wait-For-KeyPressed*$\rightarrow_{\text{alarm}}$*Error.*

In other words, an ENFA transition is the collective result of firing as many concurrent, non-conflicting, transitions as enabled by the current tape reading. Those threads where no transition fired simply remain in the same constituent TLChart state, as the case for *State*-3 in transition 2 above.

Note that conflicting simultaneously enabled ENFA transitions induce non-determinism. This is the case when *keyPressed*, *valveOpen*, and *alarm* are all true while in state configuration $\{$*Wait-For-KeyPressed*, *State*-3$\}$ i.e. when transitions 1 and 3 above are simultaneously enabled.

Bridging the gap between the modal logic based semantics of LTL and formal languages is done in the standard way using two steps, as follows. First we use finite linear model semantics for temporal logic; for example *Eventually* ρ is satisfied if there exists state *s* in the finite linear model which satisfies ρ. The second step is to translate the LTL model to an input tape for an automaton. An LTL model consists of a finite sequence of states with Boolean propositions and corresponding truth assignments assigned to each state. For example, consider a model with two states (i.e., two cycles), where $\{$*begin, ¬end, KeyPressed, ¬alarm, valveOpen*$\}$ is the truth assignment for state 0 (interpreted as cycle 0), and $\{$*¬begin, ¬end, KeyPressed, ¬alarm, valveClosed*$\}$ is the truth assignment for state 1. This model is therefore obviously exchangeable with an automaton input tape with the symbol <*begin, ¬end, KeyPressed, ¬alarm, valveOpen*> in position 0 and <*¬begin, ¬end, KeyPressed, ¬alarm, valveClosed*> in position 1. In other words, each Boolean proposition p_i and its negation $\neg p_i$ form an alphabet Σ_i. The input alphabet for the ENFA is then the Cartesian product of all Σ_i alphabets.

We now incorporate temporal conditions into ENFA behavior. First, note that every ENFA transition has a pair of propositional and temporal conditions, which are the respective conjunctions of all propositional and temporal conditions annotating its constituent TLChart transitions. Temporal conditions affect ENFA behavior via the definition of a computation. Given an input tape, a conventional one-way non-deterministic Finite Automaton (NFA) computation is essentially a sequence of "matching" transitions and corresponding tape head moves to the right; details are available in [HU]. ENFA's extend this well known definition by requiring that for every transition t_i in the computation the input tape is observed from position *i* into the future and back to the past, but without moving the tape head. The transition t_i is then enabled only if the temporal condition is satisfied by the tape, while considering position number *i* as cycle 0.

For example, using the infusion pump TLChart of Fig. 3, consider the input tape (using straight forward abbreviations of the infusion pump conditions): $\sigma=\sigma_1.\sigma_2.\sigma_3.\sigma_4.\sigma_5.\sigma_6=$
$\{IB,\neg IE,KP,\neg A,VC\}.\{\neg IB,\neg IE,\neg KP,\neg A,VC\}.$
$\{\neg IB,\neg IE,KP,\neg A,VO\}.\{\neg IB,\neg IE,KP,A,VC\}.$
$\{\neg IB,IE,\neg KP,\neg A,VC\}.\{IB,\neg IE,\neg KP,\neg A,VO\}.$

The following C_1 computation is enabled by σ; each line is considered as a cycle, starting at cycle 0:

$\{Init\}\rightarrow_{IB}$
$\{$*Wait-For-KeyPressed*,*State*-3$\}\rightarrow_{\text{(none)}}$
$\{$*Wait-For-KeyPressed*,*State*-3$\}\rightarrow_{KP,\ VO}$
$\{$*Wait-For-KeyPressed*,*State-4*$\}\rightarrow_{A}$
{Error} (a sink state)

Similarly, the following C_2 computation is also enabled by σ:

$\{Init\}\rightarrow_{IB}$
$\{$*Wait-For-KeyPressed*,*State*-3$\}\rightarrow_{\text{(none)}}$
$\{$*Wait-For-KeyPressed*,*State*-3$\}\rightarrow_{KP,\ VO}$
$\{$*Wait-For-KeyPressed*,*State-4*$\}\rightarrow_{KP,VC}$
$\{$*Wait-For-KeyPressed*,*State-3*$\}\rightarrow_{\rho}$\{Done\}

where ρ is the temporal condition *IE U IB*. ρ is enabled on cycle 4 because the input tape then points to $\sigma_5=\{\neg IB,IE,\neg KP,\neg A,VC\}$ and the tape suffix is $\sigma_5.\sigma_6=$
$\{\neg IB,IE,\neg KP,\neg A,VC\}.\{IB,\neg IE,\neg KP,\neg A,VO\}$ which satisfies ρ.

Like their logical counterpart ENFA represent assertions about a system. They do so using notation that is similar to automata, namely by accepting or rejecting strings (tapes). A classical NFA accepts a string using an existential criterion, namely, if a computation ending in a final state exists. A dual universal automaton (∀-FA) accepts a string if all computations end in a final state. Combining both acceptance criteria results in an alternating automaton. Alternatively, an existential NFA with negation can be used instead of a combination of both acceptance criteria. ENFA supports negation using (i) negation inside temporal conditions, (ii) a combination of *good* (accepting) and *error* (rejecting) states. For a given input string *s* there is one or more possible computations, some of which end in a good state while others end in an error state. Conflicts are resolved using a priority scheme where the winning computation is the computation whose last visited state configuration contains a TLChart state *St* whose priority is higher than all other TLChart states in all

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

competing configurations. If *St* is a good state then the TLChart accepts the input string otherwise the TLChart rejects it. For example, in Fig. 3 consider two computations on the input string σ, C_1 and C_2. C_1 ends in the configuration {*Error*} where the *error* state *Error* has priority 2. C_2 ends in the configuration {*Done*} where *good* state *Done* has priority 1. σ is accepted because *Done* has a higher priority than *Error*.

Whenever the priority scheme cannot resolve conflicts we arbitrarily select the error computation as overriding. Likewise, whenever a single computation ends in a configuration that contains both good and error states, then we arbitrarily select the error state as overriding.

TLCharts support two ways for specifying real-time constraints. The first way uses Harel statechart *timeout* (*tm*) events, while the second uses MTL. In Fig. 2 for example, the pair of transitions *Wait-For-KeyPressed*$\rightarrow_{tm(2min)}$*Alarm-Necessary*, and *Wait-For-KeyPressed*$\rightarrow_{keyPressed}$ *Wait-For-KeyPressed* are similar to a single transition *Wait-For-KeyPressed*$\rightarrow_{\rho}$*Alarm-Necessary* where $\rho=\Diamond_{\leq 2\text{min}}$ *keyPressed*. The two approached differ with respect to the timing in which state *Alarm-Necessary* is reached. With the first representation *Alarm-Necessary* is reached after two minutes while the second approach makes the transition immediately. We suggest a special visual delay construct, represented with thick edges, which can only be used with the following unnested temporal conditions: $\Box_{\leq d}\rho$ ($\Box\rho$ with an MTL upper bound d), $\Diamond\rho$, and $\rho U\psi$. It means that the transition is traversed only when the temporal condition becomes true, i.e., when the MTL upper bound d in for $\Box_{\leq d}\rho$ is reached, or when ψ is true in $\Diamond\psi$ or $\rho U\psi$. Hence, in Fig. 4, the transition *Done*$\rightarrow_{alarm}$*Error* is enabled only when, for the preceding transition, the *keyPressed* that satisfies *alarm U keyPressed* is detected.

From a semantic perspective, real-time measurements, used by statechart timout events and MTL constraints, are represented in our ENFA model using a standard monotonically increasing positive integer function that maps each tape cell with a real-time value.

Recall that a TLChart input string represents a sequence of combinations of stimuli and corresponding system responses; for example, the sequence σ contains *keyPressed* - generated by the environment, combined with alarm - a system generated response. Hence, from a verification standpoint, a rejected string means that the systems behavior does not comply with the specification, typically due to an incorrect system reaction to the input stimuli. This application of diagrams to specification rather than programming and design explains the existence of a sink state (the Error state), which does not typically exist in a design phase statechart.

Note that though visually similar to Harel Statecharts, TLCharts are actually used and applied more like a temporal logic specification in the following sense. TLCharts do not describe the token by token reaction of a reactive system to environment stimuli. Rather, TLCharts consider a complete input string *s*, which combines both environment inputs *s_in* and system outputs *s_out*; a TLChart asserts about the legality of an *s_out* system response to the *s_in* stimuli.

4 TLCharts with Overlapping States

The proposed automata theoretic statechart semantics described in Section 3 caters for statecharts with overlapping states [Ka]. Consider the TLChart of Fig. 4, a variant of the TLChart of Fig. 3 with overlapping states. In Fig. 4, state *State-OVLP* is an *and* state that shares its substates with the concurrent threads of state *State*-2. Fig. 4 induces a state graph that is a DAG, not a tree (syntactically illegal when considered as a pure Harel statechart). The intuitive meaning of this state overlap is that it is illegal for a key to be pressed while the valve is open.

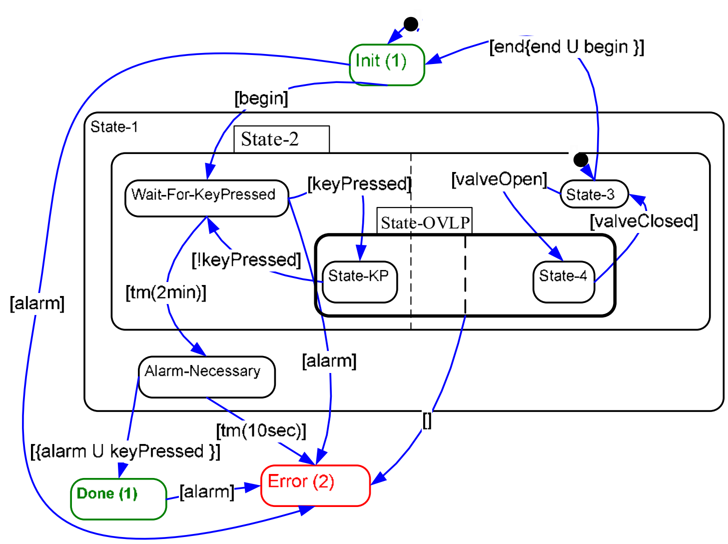

Figure 4. An extension of the TLChart of Fig. 3 with overlapping states.

From a semantics perspective, ENFA state configurations for TLCharts with overlapping states contain all state nesting information. Hence, the situation where *State-KP* and *State*-4 are simultaneously visited has two distinct possible

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

representations as ENFA state configurations: {*State*-1, *State*-2, *State-KP, State*-4}, and {*State*-1, *State-OVLP*, *State-KP, State*-4}. Therefore, the following two computations are distinct, though when considering only leaf states they look alike:

{*Init*}$\rightarrow_{IB}$
{*State*-1,*State*-2,
Wait-For-KeyPressed,*State*-3}$\rightarrow_{KP,VO}$
{*State*-1,*State*-2, *State-KP*,*State*-4}$\rightarrow_{VC}$
{*State*-1,***State*-2**, *State-KP*,*State*-3}

and

{*Init*}$\rightarrow_{IB}$
{*State*-1,*State*-2,
Wait-For-KeyPressed,*State*-3}$\rightarrow_{KP,VO}$
{*State*-1, ***State-OVLP***, *State-KP*,*State*-4}$\rightarrow_{\text{(any)}}$
{*Error*}

Given that the second computation ends in *Error*, a state with higher priority than any of the states in {*State*-1, *State*-2, *State-KP*, *State*-3}, the TLChart rejects the input, effectively stating that *State-KP* and *State*-4 cannot be visited simultaneously.

5 Armor Plating Specifications

Run time assertion checking is a common method for armor-plating programs against unexpected errors. Recently, Drusinsky suggested an armor plating method using run-time monitoring of LTL and MTL assertions combined with exception handling [D4].

TLCharts offer an opportunity for armor-plating specifications using *over-specification*, namely by adding temporal conditions to an otherwise fully specified TLChart. Consider for example requirement R1 and the corresponding TLChart of Fig. 2. A correctness property φ of interest, expressed in MTL, is that in state *Wait-For-KeyPressed*:
$(\neg\Diamond_{\leq 120}\ keyPressed) \Rightarrow$
$\Diamond_{[120,130]}\ (alarm\ U\ (keyPressed \wedge \Box\neg alarm))$.
Fig. 2,3, and 4 can be armor-plated with a transition *Wait-For-KeyPressed*->$_{\neg\varphi}$*Error*.

TLCharts appear to be a good language for armor plating due to their non-deterministic semantics.

6 NTLChart: Using Natural Language

NTLChart specifications are TLCharts where natural language snippets are used instead of temporal logic conditions. NTLCharts are based on the observation that often the temporal conditions inside TLCharts are short and use little nesting. In Fig. 3 for example, both temporal conditions have no nesting of temporal operators.

Under such circumstances, the number of possibilities for temporal conditions is rather limited. It is therefore possible to represent temporal conditions with natural language sentences using a straightforward library mapping method. For example, the sentence *alarm occurs before keyPressed* represents the less readable temporal condition ¬*keyPressed U alarm.* The Kansas State specification patterns [ACD] provides a convenient library of natural language rule snippets and corresponding temporal logic formal specifications.

7 Conclusion

Harel statechart and LTL are well-researched and advocated specification languages for reactive systems. Harel statecharts are widely popular through their UML counterpart. LTL is advocated primarily by the academia. While Harel statecharts are visual and deterministic, LTL is textual/logical and non-deterministic. TLCharts capture combine both thereby enabling specifications that are visual, partially deterministic, but also logical and non-deterministic when needed. TLCharts have a straightforward formal automata based semantics that support a meaningful interpretation of statecharts with state overlapping. With TLCharts, temporal conditions are *anchored* in states, such as *alarm U keyPressed* being anchored in the state *Alarm-Necessary* in Fig. 2. This eliminates the need to use deeply nested LTL, when using the pure LTL alternative, or to provide a fully deterministic statechart, when using the Harel statechart alternative. We call this property *just in time TL*. In addition, TLCharts enable specification armor plating.

Clearly, TLCharts can be abused; a single state TLChart with highly nested LTL and MTL conditions is a legal TLChart and so is a fully deterministic, implementation level detailed, Harel statechart. Further research is needed to establish when each constituent capability of this new formalism actually contributes a significant added value to the specification effort.

References

[ACD] G.S. Avrunin, J. C. Corbett, and M. B. Dwyer- Property Specification Patterns for Finite-State Verification, 2nd Workshop on Formal Methods in Software Practice, March 1998.

[Br] B. Bruegge- Object-Oriented Software Engineering: Conquering Complex and Changing Systems, Prentice Hall, ISBN 0-13-489725-0.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

[CPM] E. Chang, A. Pnueli- Z. Manna - Compositional Verification of Real-Time Systems, Proc. 9'th IEEE Symp. On Logic In Computer Science, 1994, pp. 458-465.

[D1] D. Drusinsky- On Synchronized Statecharts, Ph.D. Dissertation, Weizmann Institute of Science, 1988.

[D2] D. Drusinsky- Monitoring Temporal Rules Combined with Time Series, Proc. 2003 Computer Aided Verification Conference (CAV), pp. 114-117.

[D3] D. Drusinsky- Semantics and Runtime Monitoring of TLCharts: Statechart Automata with Temporal Logic Conditioned Transitions. Proc. 4'th Runtime Verification workshop, RV'04 (invited).

[D4] D. Drusinsky- Specs Can Handle Exceptions. Embedded Developers Journal, November 2001, pp. 10-14. (http://eet.com/embedsub/archive.html).

[DH] D. Drusinsky and David Harel. On the power of bounded concurrency I: Finite Automata. Journal of the ACM, 41(3): 517-539, May 1994.

[Ha] D. Harel- Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming 8, pp. 231-274, 1987

[HN] D. Harel and A. Naamad- The Statemate Semantics of Statecharts. ACM Tran. of Software Engineering and Methodology, 5(4) Oct 1996.

[EFHL] C. Eisner, D. Fishman, J. Havlicek, Y. Lustig, A. McIsaac, D. Van Campenhout-Reasoning with Temporal Logic on Truncated Paths, Proc. 2003 Computer Aided Verification Conference (CAV), pp. 27-39.

[E] M.Enciso, I. P. de Guzm´an, C.Rossi- Using Temporal Logic to represent Dynamic Behaviour of UML Statecharts, en: ECOOP 2002 Workshop on Integration and Transformation of UML Models (2002).

[HU] J. Hopcroft. and J. Ullman- Theory of Formal Languages and Automata, Addison Wesley, 2'nd edition, 2001, ISBN 0-201-44124-

[Ka] Kahanna, C.A.- Statecharts with overlapping states, M.S. Thesis, Dept. of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel, 1986 (Hebrew).

[MP] Z. Manna, A. Pnueli - Verification of Concurrent Programs: Temporal Proof Principles, Proc. of the Workshop on Logics of Programs, Springer LNCS, 1981 pp. 200-252.

[Pn] A. Pnueli - The Temporal Logic of Programs, Proc. 18'th IEEE Symp. on Foundations of Computer Science, 1977, 46-57.

[RB] J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, W. Lorensen - Object Oriented Modeling and Design, Prentice Hall, ISBN 0-13-629841-9.

[SR] S. R. Sowmya and S. Ramesh - Extending Statecharts with Temporal Logic, IEEE Transactions on Software Engineering, Vol. 24, No. 3, March 1998 – 1998.