Innovations in Systems and Software Engineering manuscript No.
(will be inserted by the editor)

Ji Zhang · Zhinan Zhou · Betty H.C. Cheng* · Philip K. McKinley

Specifying Real-Time Properties in Autonomic Systems

Received: date / Accepted: date

Abstract Increasingly, computer software must adapt dynamically to changing conditions. The correctness of adaptation cannot be rigorously addressed without precisely specifying the requirements for adaptation. In many situations, these requirements involve absolute time, in addition to a logical ordering of events. This paper introduces an approach to formally specifying such timing requirements for adaptive software. We introduce TA-LTL, a timed adaptation-based extension to linear temporal logic, and use this logic to specify three timing properties associated with the adaptation process: *safety*, *liveness*, and *stability*. A dynamic adaptation scenario involving interactive audio streaming software is used to illustrate the timed temporal logic.

Keywords Autonomic Systems · Adaptation · Timing Properties · Temporal Logic · Model Checking

1 Introduction

Interests in adaptive computing systems have grown dramatically in the past few years [23]. One of several driving factors is the increasing demand for *autonomic computing* [15], which refers to systems being capable of managing and protecting their own resources with only high-level human guidance. Adaptive software behavior, including the dynamic recomposition of the code itself, has been studied extensively with respect to different system layers [23]. Some projects have addressed adaptation at the operating system level [4; 7], while others have focused on middleware [16; 25; 31]. In addition to its traditional role in hiding resource distribution and platform heterogeneity, middleware can be used to address several concerns related to adaptive behavior, including quality-of-service, fault tolerance, and security.

Dynamically adaptive software can be more resilient to changing conditions and offer higher availability than nonadaptive software [23]. The design of adaptive software, however, must be supported by a programming paradigm that lends itself to automated checking of both functional and nonfunctional properties of the system. In particular, correctness of adaptive software systems cannot be rigorously addressed without precisely specifying their requirements [6]. Many types of adaptation involve timing requirements, including requirements on the allowable sequences of actions in reconfiguring code, and absolute timing requirements arising from real-time characteristics of the application. This paper focuses on the specification of timing requirements associated with dynamic software adaptation.

Previously, Zhang and Cheng proposed A-LTL [34; 35], an extension to linear temporal logic for formally specifying the semantics of adaptive and recomposable software. A-LTL enables software developers to explicitly specify adaptive behavior. Moreover, A-LTL enables application of automated analysis methods to adaptive software, including the use of model checking to verify the correctness of a model, and dynamic insertion of adaptation logic code into the software [36]. However, A-LTL specifies only the relative temporal ordering among events and system states that occur during the adaptation process. While this capability is suited to specifying many types of adaptive behavior, A-LTL is insufficient to specify systems with real-time requirements, where the absolute timing may play an important role.

Ji Zhang · Zhinan Zhou · Betty H.C. Cheng · Philip K. McKinley
Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824, USA
Tel.: +1 517-355-8344
Fax: +1 517-432-1061
E-mail: {zhangji9, zhouzhin, chengb, mckinley}@cse.msu.edu
Present address: Zhinan Zhou
San Jose Mobile Communications Lab
Samsung Telecommunications America
85 W. Tasman Dr.
San Jose, CA 95134
Tel.: +1 408-544-5160
E-mail: z.zhou@samsung.com

In this paper we introduce *Timed* A-LTL (TA-LTL), an extension to A-LTL that includes a metric for time [5; 41]. Specifically, TA-LTL enables temporal constraints to be expressed in a quantitative form, in addition to specifying relative temporal ordering. We model an adaptive software program as the composition of a finite number of steady-state programs [2; 35] and the adaptations among these steady-state programs. We assume that the properties of each program have already been specified with a Linear Temporal Logic (LTL) formula. Furthermore, we assume the adaptation process from one program to another has been specified with an A-LTL formula. TA-LTL enables us to specify three types of properties of adaptation that involve absolute time, namely the *safety*, *liveness*, and *stability* properties. We have successfully applied this specification technique to a number of prototype adaptive programs, including adaptive communication software components.

The remainder of this paper is organized as follows. Section 2 provides background information on adaptive communication services, and Section 3 discusses related work on specifying temporal constraints. In Section 4 we introduce the syntax and semantics of TA-LTL, and Section 5 describes the use of TA-LTL in specifying the timing properties of three commonly used adaptation semantics. Section 6 illustrates the approach with a mobile communication example. Section 7 discusses some issues regarding TA-LTL, including its expressiveness, decidability, model checking algorithms, and asynchronous extensions. Section 8 concludes the paper and discusses future directions.

2 Background

This study is part of *RAPIDware* [1], a project sponsored by the U.S. Office of Naval Research. RAPIDware addresses the design and evaluation of adaptive middleware to support interactive applications in dynamic, heterogeneous environments. The project investigates software technologies and programming abstractions for building adaptive systems. An overarching premise of the project is that the design of adaptive software must be grounded in principled, rigorous software engineering methods in order to ensure that the integrity of the system is not compromised as the system adapts, and that the system can be effectively tested and maintained. In this paper, we introduce TA-LTL as part of that effort.

While TA-LTL is applicable to many types of software systems, our initial focus, described here, is on adaptive communication services. Responding to changes in the environment is particularly important in mobile computing, since conditions on wireless channels are highly dynamic and devices must try to maintain an acceptable quality of service. Dynamic adaptation can be used to enhance fault tolerance, improve quality of service, and accommodate changing security policies as users roam among wireless domains. However, reconfiguration of communication software can introduce timing problems, particularly when the data stream has real-time requirements.

After presenting the design and use of TA-LTL, we describe a case study in which we specify temporal constraints on *MetaSockets* [30], an adaptive communication component developed previously in the RAPIDware project. MetaSockets provide the same imperative functionality as regular Java sockets, including methods for sending and receiving data. However, their internal structure and behavior can be adapted at run time in response to changes in their environment. Figure 1 illustrates the internal architecture of the particular type of MetaSockets used in this study. Packets are passed through a pipeline of *Filter* components, each of which processes the packets. Example filter services include: auditing traffic and usage patterns, transcoding data streams into lower-bandwidth versions, encrypting and decrypting data, and using forward error correction (FEC) to make data streams more resilient to packet loss. Figure 1 shows that the MetaSocket also supports special types of methods to insert and remove filters, as well as retrieve their status.

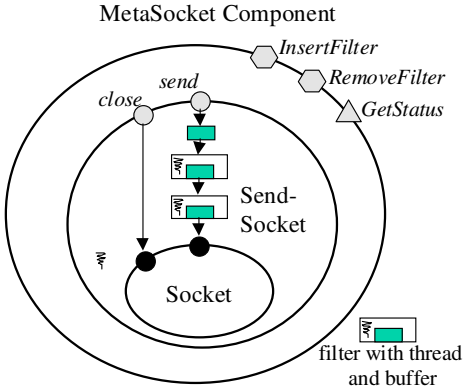


Fig. 1 Structure of a MetaSocket [30].

An earlier experimental study [40] demonstrated the effectiveness of MetaSockets in supporting interactive audio streaming to mobile devices under dynamic wireless channel conditions. In particular, MetaSockets enable a system to invoke stronger error control methods under conditions of high packet loss, as well as implement energy conservation strategies that extend battery lifetime. However, that study also revealed that certain MetaSockets configurations can exhibit processing delays large enough to produce jitter in the delivery and playback of the audio stream. In the case study presented later, we demonstrate how TA-LTL specifications can be used to prevent such situations.

3 Related Work

In the past decade, researchers have proposed many techniques, tools, and models for the formal specification of real-time systems. Formal specification languages typically enumerate the temporal constraints among events and actions, such as periodicity, invariants, liveness, and safety conditions. For example, Linear Temporal Logic (LTL), introduced by Pnueli [26], is a temporal extension to the propositional logic. The temporal connectives of LTL, including $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (eventually), $\Box$ (always), etc. describe the temporal relationships between states that characterize the temporal system evolution.

However, LTL does not provide a metric of time and describes only temporal ordering relationships between states. Several approaches have been proposed to introduce time explicitly to LTL, providing the capability to make quantitative temporal assertions. Real-Time Temporal Logic (RTTL) [24] extends LTL with real-time operations and proof rules for real-time properties. Real-Time Logic (RTL) [14] extends first order logic with a set of elements for specifying real-time system requirements by presenting an absolute clock to measure time progression. Metric Temporal Logic (MTL) [17] extends first order logic with temporal operators, and includes a metric for time to describe the structure of the temporal domain. In addition to the proposition values, Timed Propositional Temporal Logic (TPTL) [3] has a single clock reading at every state. Each clock reading can be "frozen" with the use of a freeze operator so that it can be compared with others. All these formal specification languages have been shown to be effective in specifying certain types of real-time properties. However, to our knowledge, none has been applied to specifying dynamic software adaptation, likely due to the lack of expressiveness for capturing adaptation semantics or the complexity of the syntax needed to precisely specify adaptation semantics.

Other work has addressed the correctness of adaptations. For example, Kulkarni *et al.* [20] introduced a transitional-invariant lattice approach; theorem proving is used to show that during and after an adaptation, the adaptive program is in correct states with respect to satisfying the transitional invariants. Other approaches [11; 19] use formal languages to describe the structural changes of the software at the design and implementation levels. Finally, protocols and algorithms during component replacement have been proposed [4; 10; 18; 39] to guarantee safe adaptation. However, these methods do not explicitly address timing requirements of the adaptation procedure.

In this paper, we address this issue by proposing a language to specify the timing properties of adaptation behavior in autonomic systems. We extend our previous work on using A-LTL to specify adaptation semantics by introducing a metric for time.

4 Timed Adapt Operator-Extended LTL

The TA-LTL language is based on two existing temporal logics: A-LTL [34] and TPTL [3]. We first briefly review both logics, and then give detailed syntax and semantics of TA-LTL.

4.1 Supporting Temporal Logics

A-LTL. To specify adaptation semantics, we have proposed A-LTL (Adapt operator-extended LTL) [34], an extension to LTL. With A-LTL, the underlying program model is a state machine that changes its behavior during its execution. The state space describing each different kind of behavior is a *steady-state program*. The state space describing the change of behavior from one non-adaptive program (*source program*) to another (*target program*) is a simple adaptive program, which can be identified by the source and target program pair. In general, an n-plex adaptive program comprises n steady-state programs.

To specify adaptation behavior, A-LTL extends LTL with the adapt operator ($\stackrel{\Omega}{\multimap}$). Among existing temporal connectives, the closest to the adapt operator is the *chop operator* (";") from the *choppy logic* introduced by Rosner and Pnueli [29], and the ITL introduced by Bowman and Thompson [9]. Informally, a software program satisfying "$\phi \stackrel{\Omega}{\multimap} \psi$" (read as ϕ adapts to ψ with adaptation constraint Ω) means that the program initially satisfies ϕ, and at a certain state A, it fulfills all the obligations demanded by ϕ and stops being constrained by ϕ, and in the next state B, starts to satisfy ψ, where ϕ and ψ are two temporal logic formulae. The state sequence (A, B) satisfies Ω, where Ω is an LTL formula evaluated on a sequence of two states. Note that LTL semantics is defined over infinite sequences. The semantics of both ϕ and Ω, however, has been redefined to be applied over sequences of finite length as described below.

We have defined A-LTL semantics on both finite state sequences (denoted by "$\models_{fin}$") and infinite state sequences (denoted by "$\models_{inf}$").

- Operators ($\rightarrow$, $\wedge$, $\vee$, $\Box$, $\Diamond$, $\mathcal{U}$, $\neg$, etc) are defined similarly as those in LTL.
- If σ is an infinite state sequence and ϕ is an LTL formula, then σ satisfies ϕ in A-LTL if and only if σ satisfies ϕ in LTL. Formally, $\sigma \models_{inf} \phi$ iff $\sigma \models \phi$ in LTL.
- If σ is a finite state sequence and ϕ is an A-LTL formula, then $\sigma \models_{fin} \phi$ if and only if $\sigma' \models_{inf} \phi$, where σ' is the infinite state sequence constructed by repeating the last state of σ.

– $\sigma \models_{inf} \phi \overset{\Omega}{\frown} \psi$ if and only if there exists a finite state sequence $\sigma' = (s_0, s_1, \cdots s_k)$ and an infinite state sequence $\sigma'' = (s_{k+1}, s_{k+2}, \cdots)$, such that

$$\begin{aligned} (s_k, s_{k+1}) &\models_{fin} \Omega, \\ \sigma' &\models_{fin} \phi, \\ \sigma'' &\models_{inf} \psi, \\ and, \sigma &= \sigma' \frown \sigma'' \end{aligned}$$

where ϕ, ψ, and Ω are A-LTL formulae, and "$\frown$" is the sequence concatenation operator. Informally, a sequence satisfying $\phi \overset{\Omega}{\frown} \psi$ can be considered the concatenation of two subsequences, where the first subsequence satisfies ϕ, the second subsequence satisfies ψ, and the two states connecting the two subsequences satisfy Ω.

In practice, we have found A-LTL to be more convenient than LTL in specifying various adaptation semantics [37]. In theory, we can prove that while A-LTL and LTL have the same expressive power, A-LTL is at least exponentially more succinct than LTL in specifying adaptation properties [36].[1]

TPTL. Temporal logics with a metric for time allow the definition of quantitative temporal relationships (in addition to qualitative ordering), such as distance among events and durations of events. A typical way to add a metric for time in a propositional temporal logic is to replace the unrestricted temporal operators by time-bounded operators. For example, the bounded operator $\Diamond_{[1,3]}\phi$ states "ϕ will become true eventually within 1 to 3 time units from the current time."

An alternative way to represent time quantitatively is to use the *freeze quantification* proposed by Alur and Henzinger [3] in TPTL (Timed Propositional Temporal Logic). TPTL is a formalism for specifying real-time properties. In TPTL, a time variable x can be bound by a freeze quantifier ("$x.$"), which freezes x to the time the temporal context is evaluated. Formally, given a supply of variables $V = \{x, y, z \cdots\}$, a set of atomic propositions $P = \{p, q, \cdots\}$, TPTL formulae are inductively defined as

$$\begin{aligned} &\pi := x + c \mid c \\ &\phi := p \mid \pi_1 \leq \pi_1 \mid \pi_1 \equiv_d \pi_2 \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc \phi \mid \\ &\qquad \phi_1 \, \mathcal{U} \, \phi_2 \mid x.\phi \end{aligned}$$

for $x \in V$, $p \in P$, and $c, d \in \mathbb{N}$, $d \neq 0$.

In TPTL, the timing constraints are of the form $\pi_1 \leq \pi_1$, $\pi_1 \equiv_d \pi_2$. The "$\leq$" operator is the traditional arithmetic inequality operator, and $\pi_1 \equiv_d \pi_2$ means that π_1 is congruent to π_2 modulo the constant d [3]. More notably, the formula $x.\phi$ means that ϕ is *true* if all occurrences of x in ϕ are replaced by the current time τ_0. Other propositional and temporal operators are defined as usual in LTL.

4.2 Syntax of TA-LTL

TA-LTL is intended to address the *safety*, *liveness*, and *stability* of adaptation behaviors. Safety and liveness are adaptation properties to restrict the duration of adaptation state transitions. Stability constrains the frequency of state changes to ensure a system does not "thrash" by oscillating between states. To support real-time in A-LTL, we adopt the real-time metric in TPTL and apply it to formulae in A-LTL.

The formulae of TA-LTL are constructed from proposition symbols and timing constraints comprising inequality operators, temporal operators, and time reading variables. Let P be a set of proposition symbols ($p, q, r, \ldots$), let N be the set of nonnegative integers, and let V be an infinite supply of time reading variables ($x, y, z, \ldots$). The terms of time π and formulae ϕ of TA-LTL are inductively defined as follows:

$$\begin{aligned} &\pi := x + c \mid c \\ &\phi := p \mid \pi_1 \sim \pi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc \phi \mid \\ &\qquad \Box\phi \mid \Diamond\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \overset{\Omega}{\frown} \phi_2 \mid x.\phi \end{aligned}$$

for $x \in V$, $p \in P$, and integer constant $c \in N$.

In TA-LTL, the timing constraints are of the form $\pi_1 \sim \pi_2$, where π_1 and π_2 are two terms, and $\sim$ represents one of the arithmetic inequality operators $<, \leq, >, \geq, =$.

The temporal operators $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (finally), and $\Box$ (always) are similar to the temporal operators defined in LTL. The operator $\overset{\Omega}{\frown}$ is similar to the adapt operator defined in A-LTL. "$x.$" is similar to the freeze operator in TPTL, and other propositional operators, such as $\vee$ (or), $\rightarrow$ (imply), $\leftrightarrow$ (co-imply), are defined similarly to those in propositional logic.

4.3 Semantics of TA-LTL

TA-LTL is evaluated over *timed state sequences*, where each state is an interpretation of a subset of P. A timed

[1] It is a common misunderstanding that the *adapt* operator in A-LTL can be simply replaced by the *next* or *until* operator in LTL. We have proved [36] that it is not the case.

state sequence is an infinite sequence of states, each of which is labeled with a discrete time t ($t \in \mathbb{N}$) [3]. Formally, a *state sequence* σ

$$\sigma = \sigma_0\sigma_1\sigma_2\ldots$$

is an infinite sequence of states, where $\sigma_i \subseteq P$ for all $i \geq 0$. A *time sequence*

$$\tau = \tau_0\tau_1\tau_2\ldots$$

is an infinite sequence of time readings, where $\tau_i \in \mathbb{N}$ for all $i \geq 0$, and τ satisfies

Monotonicity: $\tau_i \leq \tau_{i+1}$ for all $i \geq 0$;

Progress: For all $t \in \mathbb{N}$, there is some i ($i \geq 0$) such that $\tau_i > t$.

Thus, a timed state sequence is denoted as a pair $\rho = (\sigma, \tau)$, or it can also be considered as an infinite sequence of pairs $\rho = (\sigma_0, \tau_0), (\sigma_1, \tau_1), \ldots$. We use ρ^i to represent the i^{th} suffix of ρ, i.e., $\rho^i = \rho_i, \rho_{i+1}, \cdots$.

An *environment* ε is an assignment of all the variables in V, which is also extended to be applied to terms. For example, $\varepsilon(x + c)$ is defined to be $\varepsilon(x) + c$. We use $\rho \models_\varepsilon \phi$ to represent the satisfaction relationship between a timed state sequence ρ and a TA-LTL formula ϕ under the environment ε. Similar to A-LTL, we also define TA-LTL semantics in the domain of timed state sequences with finite length. We use $\lambda \models_{fin\varepsilon} \phi$ to represent that a finite timed state sequence λ *satisfies* TA-LTL formula ϕ in the finite sequence domain. The TA-LTL semantics is formally defined as follows:

- $\lambda \models_{fin\varepsilon} \phi$ if and only if $\rho \models_\varepsilon \phi$, where ρ is the infinite timed state sequence obtained by extending λ with its last state infinite times. This rule related the semantics in infinite and finite timed state sequence domains.
- $\rho \models_\varepsilon p$ for all $p \in P$, if and only if $p \in \sigma_0$.
- $\rho \models_\varepsilon \pi_1 \sim \pi_2$ if and only if $\varepsilon(\pi_1) \sim \varepsilon(\pi_2)$.
- $\rho \models_\varepsilon \phi \wedge \psi$ if and only if $\rho \models_\varepsilon \phi$ and $\rho \models_\varepsilon \psi$.
- $\rho \models_\varepsilon \neg\phi$ if and only if $\rho \not\models_\varepsilon \phi$.
- $\rho \models_\varepsilon \bigcirc\phi$ if and only if $\rho^1 \models_\varepsilon \phi$.
- $\rho \models_\varepsilon \phi\,\mathcal{U}\,\psi$ if and only if exists i ($i \leq 0$) such that $\rho^i \models_\varepsilon \psi$, and for all j ($0 < j < i$), $\rho^j \models_\varepsilon \phi$.
- $\rho \models_\varepsilon x.\phi$ if and only if $\rho \models_{\varepsilon[x:=\tau_0]} \phi$, where $\varepsilon[x := \tau_0]$ is the environment derived by replacing the assignment of x in ε with τ_0.
- $\rho \models_\varepsilon \phi \xrightarrow{\Omega} \psi$ if and only if exists i ($i > 0$) such that $\rho^i \models_\varepsilon \psi$ and $\rho_0, \cdots \rho_{i-1} \models_{fin\varepsilon} \phi$.
- Other operators ($\vee$,$\rightarrow$, $\Diamond$, $\Box$, etc) are defined similarly as those used in LTL.

In this paper, we assume all TA-LTL formulae are closed, i.e., any occurrence of time variable x must be within the scope of a freeze operator "$x.$".

5 Specifying Adaptation Timing Properties

Figure 2 depicts three commonly used adaptation scenarios [34]: Figure 2(a) depicts the *one-point adaptation* (the source program adapts to the target program at a certain point during its execution); Figure 2(b) shows the *guided adaptation* (the source program adapts to the target program with a restriction condition); and Figure 2(c) depicts the *overlap adaptation* (under the restriction conditions, the target program behavior starts before the source program behavior stops). There are many other possible adaptation semantics. In this paper, however, we focus on specifying the timing properties of these adaptation semantics with TA-LTL by binding time variables to the temporal formulae. Specifically, we investigate three real-time properties of adaptation semantics: *safety*, *liveness*, and *stability*.

Safety asserts that properties that may invalidate the adaptation will not happen. The timing properties of safety enforce that the source/target program should enter the restriction condition or safe state within a certain time period after receiving the adaptation request. If this time constraint cannot be satisfied, then it might be too late for the system to react to the dynamically changing environment, attacks, user requirements, etc. Safety properties are most important to adaptations when responding to security threats in which late responses may cause irreversible damage to the system.

Liveness asserts properties that respond to the adaptation requests and asserts that the adaptation goals will eventually be satisfied in time. The liveness timing properties restrict the time periods allowed for the source program to adapt to the target program upon receiving adaptation requests. If this time constraint cannot be satisfied, then the adaptation needs may become invalid or the adaptation may cause the system to enter an inconsistent state [38; 39].

Stability asserts how long the system should remain in certain states. In order to avoid adaptation oscillation caused by frequent adaptation triggering, the timing properties of stability constrain the time interval between two successive adaptations and thus ensure that the system executes in a steady-state program for a certain amount of time before the next adaptation is allowed. If the stability time constraint is violated, then the system might enter an unstable state.

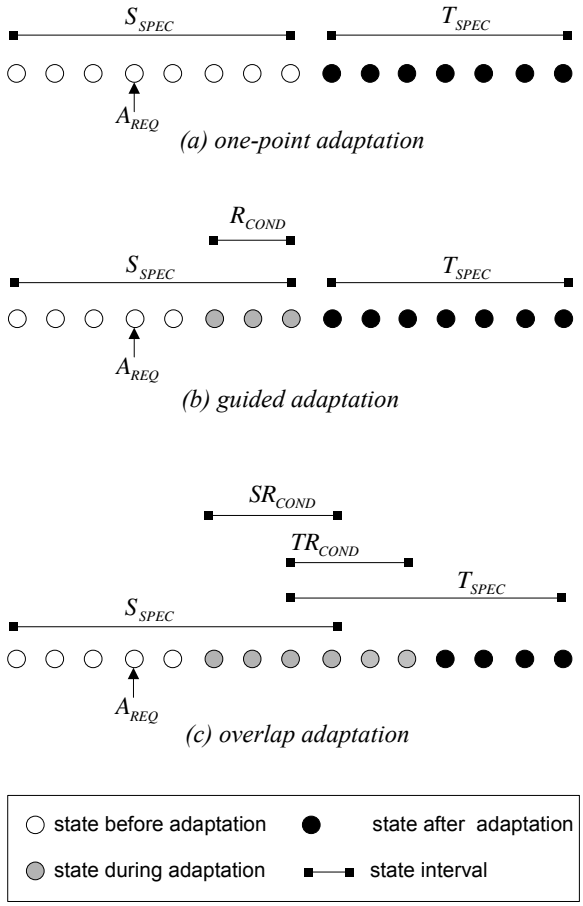


Fig. 2 Three adaptation scenarios.

While each steady-state program may have its own real-time constraints, in this paper, we only focus on adaptation related real-time constraints and assume that the source and the target steady-state programs have been both specified in LTL. We refer to these specifications as *base specifications* and denote them as S_{SPEC} and T_{SPEC}, respectively. We specify the adaptation from the source program to the target program with A-LTL by extending the base specifications of the source and the target programs. For some adaptations, the source/target program behavior may need to be constrained during the adaptation. We use an LTL formula to specify the *restriction condition*, R_{COND}, during the adaptation.

We use a proposition A_{REQ} to indicate the receipt of an adaptation request to a target program. We identify the *quiescent states* for adaptation to be those states where all the obligations required by the source specification S_{SPEC} are fulfilled by the source program, and thus it is safe to terminate the source behavior.

5.1 One-Point Adaptation

Under one-point adaptation semantics, after receiving an adaptation request, the source program adapts to the target program at a certain point during its execution. The prerequisite is that the source program should always eventually reach quiescent states during its execution. This semantics is depicted in Figure 2(a), where circles represent a sequence of states. Solid lines represent state intervals and the label of each solid line represents the property that is held by the interval. The arrow points to the states in which an adaptation request is received. This adaptation semantics can be specified by A-LTL as follows:

$$SPEC_{ONE\text{-}POINT} = (S_{SPEC} \wedge \Diamond A_{REQ}) \overset{\Omega}{\rightarrow} T_{SPEC} \quad (1)$$

This formula states that the program initially satisfies S_{SPEC}. When the program reaches a quiescent state, i.e., all obligations demanded by S_{SPEC} are fulfilled, the program stops being obligated by S_{SPEC} and starts to satisfy T_{SPEC}. In the context of adaptive processing of data streams, an example is to insert the encoder at the sender side where each operation is applied independently to each packet in the outgoing stream.

We specify the liveness timing constraints for one-point adaptation as follows:

$$LIVE_{ONE\text{-}POINT} = \Diamond x.(A_{REQ} \overset{\Omega}{\rightarrow} y.T_{SPEC} \wedge (y \leq x + t_{live})) \quad (2)$$

The formula states that once the source program receives an adaptation request (specified by A_{REQ}), it should adapt to the target program (specified by T_{SPEC}) within t_{live}, where t_{live} represents the upper bound of the constrained time frame.

The stability timing constraint for one-point adaptation is specified as follows:

$$STABLE_{ONE\text{-}POINT} = x.((S_{SPEC} \wedge \Diamond A_{REQ}) \overset{\Omega}{\rightarrow} y.T_{SPEC} \wedge (y \geq x + t_{freq})) \quad (3)$$

The formula states that the source program should continue to satisfy S_{SPEC} for at least t_{freq} before it can start to satisfy T_{SPEC}.

The timing constraints for one-point adaptation are illustrated in Figure 3, where the time difference constrained by each formula is denoted by Δt.

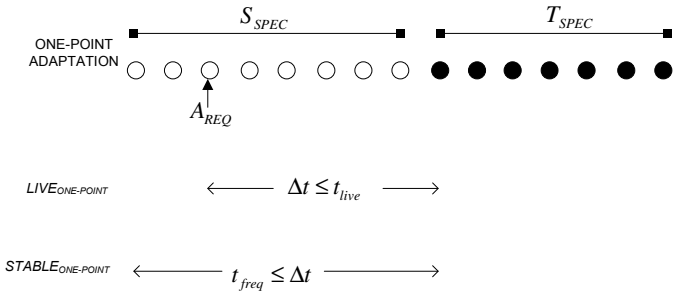


Fig. 3 Real-time constraints for one-point adaptation.

5.2 Guided Adaptation

Under guided adaptation semantics (visually depicted in Figure 2(b)), after receiving an adaptation request, the system first applies a restriction condition to the source program behavior, and then adapts to the target program when it reaches a quiescent state. This semantics is suited for adaptations where source programs cannot otherwise guarantee reaching a quiescent state within a given amount of time. The restriction condition ensures that the source program will reach a quiescent state. This situation arises when the system needs to provide continued service of the old system before switching to the new system. For example, if we are applying encryption encoding/decoding to a sender and receiver, the receiver needs to process any unencoded packets that are in-flight or buffered at the receiver, before inserting the decoder into the processing chain. This adaptation semantics can be specified by A-LTL as follows:

$$\begin{aligned} SPEC_{GUIDED} = {} & \big(S_{SPEC} \wedge (\Diamond A_{REQ} \\ & \xrightarrow{\Omega_1} R_{COND})\big) \\ & \xrightarrow{\Omega_2} T_{SPEC} \end{aligned} \tag{4}$$

The specification states that initially S_{SPEC} is satisfied. After an adaptation request, A_{REQ}, is received, the program should satisfy a restriction condition R_{COND} (marked with $\xrightarrow{\Omega_1}$). When the program reaches a quiescent state of the source, the program stops being constrained by S_{SPEC}, and starts to satisfy T_{SPEC} (marked with $\xrightarrow{\Omega_2}$). The *hot-swapping* technique introduced by Appavoo *et al* [4] and the safe adaptation protocol [38; 39] introduced in our previous work use the guided adaptation semantics.

The safety timing property of $SPEC_{GUIDED}$ can be specified by TA-LTL as follows:

$$\begin{aligned} SAFE_{GUIDED} = {} & \Diamond x.(A_{REQ} \\ & \xrightarrow{\Omega_1} y.R_{COND} \wedge (x + t_{safe} \geq y)) \\ & \xrightarrow{\Omega_2} T_{SPEC} \end{aligned} \tag{5}$$

This formula states that upon receiving the adaptation request, the source program should satisfy the restriction condition, R_{COND}, within t_{safe}, otherwise the response is too late to ensure safety in the system, where t_{safe} represents the upper bound of the constrained time frame.

The liveness property $LIVE_{GUIDED}$ for the guided adaptation is specified as follows:

$$\begin{aligned} LIVE_{GUIDED} = {} & \Diamond x.(A_{REQ} \\ & \xrightarrow{\Omega_1} R_{COND} \\ & \xrightarrow{\Omega_2} y.T_{SPEC} \wedge (y \leq x + t_{live})) \end{aligned} \tag{6}$$

This formula has a similar meaning to the liveness property for the one-point adaptation. It states that once the source program receives an adaptation request (specified by A_{REQ}), it should adapt to the target program (specified by T_{SPEC}) within t_{live}, where t_{live} represents the upper bound of the constrained time frame.

The stability property $STABLE_{GUIDED}$ for the guided adaptation is as follows:

$$\begin{aligned} STABLE_{GUIDED} = {} & x.(S_{SPEC} \wedge (\Diamond A_{REQ} \\ & \xrightarrow{\Omega_1} R_{COND}) \\ & \xrightarrow{\Omega_2} y.T_{SPEC} \wedge (y \geq x + t_{freq})) \end{aligned} \tag{7}$$

The formula states that the source program should continue to satisfy S_{SPEC} for at least t_{freq} before it can start to satisfy T_{SPEC}.

The real-time constraints for guided adaptation are illustrated in Figure 4.

5.3 Overlap Adaptation

Under overlap adaptation semantics, depicted in Figure 2(c), the target steady-state program behavior starts before the source steady-state program behavior stops. During the overlap of the source and the target behavior, a restriction condition, R_{COND}, is applied to safeguard the correct behavior of the program. This adaptation semantics is suited for the case when continuous service from the adaptive program is required. The restriction condition ensures that the source program reaches a quiescent state. For example, in an adaptive communication program, a receiver may need to be able to handle both

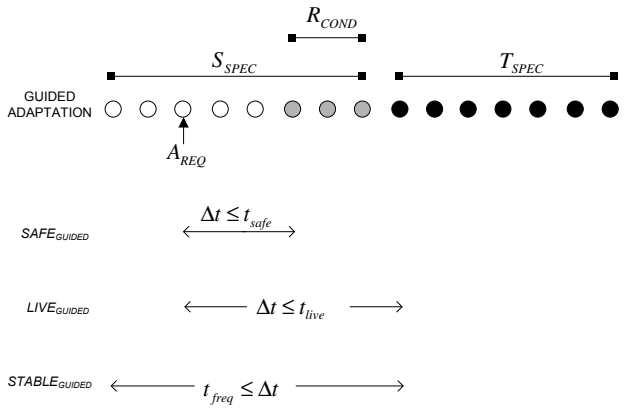


Fig. 4 Real-time constraints for guided adaptation.

unencoded and encoded packets for a period of time after the filter has been reconfigured. The overlap adaptation semantics is formally specified in A-LTL as follows:

$$\begin{aligned} SPEC_{OVERLAP} = (&S_{SPEC} \wedge (\Diamond A_{REQ} \\ &\xrightarrow{\Omega_1} SR_{COND}) \\ &\xrightarrow{\Omega_2} true) \\ &\wedge (\Diamond A_{REQ} \\ &\xrightarrow{\Omega_3} T_{SPEC} \wedge (TR_{COND} \\ &\xrightarrow{\Omega_4} true)) \end{aligned} \tag{8}$$

This formula states that the entire adaptation process comprises two parts. Initially only the S_{SPEC} is satisfied. After an adaptation request, A_{REQ}, is received, the system should start to satisfy a source restriction condition, SR_{COND} (marked with $\xrightarrow{\Omega_1}$). When the system reaches a quiescent state of the source program, the adaptive program stops being obliged by S_{SPEC} and SR_{COND} (marked with $\xrightarrow{\Omega_2}$). The second part of the formula is used to specify the target program behavior which starts to satisfy T_{SPEC} and target restriction condition TR_{COND} upon receiving adaptation request A_{REQ} (marked with $\xrightarrow{\Omega_3}$). Finally only the T_{SPEC} continue to be satisfied (marked with $\xrightarrow{\Omega_4}$). The *graceful adaptation protocol* introduced by Chen *et al* [10] and the distributed reset protocol introduced by Kulkarni *et al* [21] use the overlap adaptation semantics.

The real-time properties of $SPEC_{OVERLAP}$ are shown in Formula 9–14 below:

$$\begin{aligned} SAFE_{REQ\text{-}ADAPT} = \Diamond x.(&A_{REQ} \\ &\xrightarrow{\Omega_1} y.SR_{COND} \wedge (x + t_{s-ra} \ge y)) \\ &\xrightarrow{\Omega_2} true \end{aligned} \tag{9}$$

$$\begin{aligned} LIVE_{REQ\text{-}FULFILL} = \Diamond x.(&A_{REQ} \\ &\xrightarrow{\Omega_1} SR_{COND} \\ &\xrightarrow{\Omega_2} y.(x + t_{l-rf} \ge y)) \end{aligned} \tag{10}$$

$$\begin{aligned} LIVE_{REQ\text{-}TSTART} = \Diamond x.(&A_{REQ} \\ &\xrightarrow{\Omega_3} y.(T_{SPEC} \wedge TR_{COND} \\ &\xrightarrow{\Omega_4} (x + t_{l-rt} \ge y))) \end{aligned} \tag{11}$$

$$\begin{aligned} LIVE_{TSTART\text{-}COMPLETE} = \Diamond &A_{REQ} \\ &\xrightarrow{\Omega_3} (x.T_{SPEC} \wedge (TR_{COND} \\ &\xrightarrow{\Omega_4} y.(x + t_{l-tc} \ge y))) \end{aligned} \tag{12}$$

$$\begin{aligned} LIVE_{REQ\text{-}COMPLETE} = \Diamond x.(&A_{REQ} \\ &\xrightarrow{\Omega_3} T_{SPEC} \wedge (TR_{COND} \\ &\xrightarrow{\Omega_4} y.(x + t_{l-rc} \ge y))) \end{aligned} \tag{13}$$

$$\begin{aligned} STABLE_{OVERLAP} = x.(&S_{SPEC} \wedge \Diamond A_{REQ} \\ &\xrightarrow{\Omega_1} SR_{COND} \\ &\xrightarrow{\Omega_2} y.(x + t_{freq} \le y)) \end{aligned} \tag{14}$$

In Formula 9, $SAFE_{REQ\text{-}ADAPT}$ states that upon receiving the adaptation request, the source program should satisfy the source restriction condition SR_{COND} within t_{s-ra}, otherwise the response time is too long to be effective. In Formula 10, $LIVE_{REQ\text{-}FULFILL}$ states that once the source program receives an adaptation request, it should complete the whole adaptation process within t_{l-rf}. Similarly, in Formula 11, $LIVE_{REQ\text{-}TSTART}$ indicates that once the adaptation request is received, the target program must start within t_{l-rt}. In Formula 12, $LIVE_{TSTART\text{-}COMPLETE}$ states that once the target behavior starts, the entire adaptation process must complete with t_{l-tc}. In Formula 13, $LIVE_{REQ\text{-}COMPLETE}$ states the time from receiving the adaptation request to being fully functioning must be within t_{l-rc}. And finally, in Formula 14, $STABLE_{OVERLAP}$ states that the source program should continue to satisfy S_{SPEC} for at least t_{freq} before it may stop satisfying S_{SPEC}. The real-time constraints for overlap adaptation are illustrated in Figure 5.

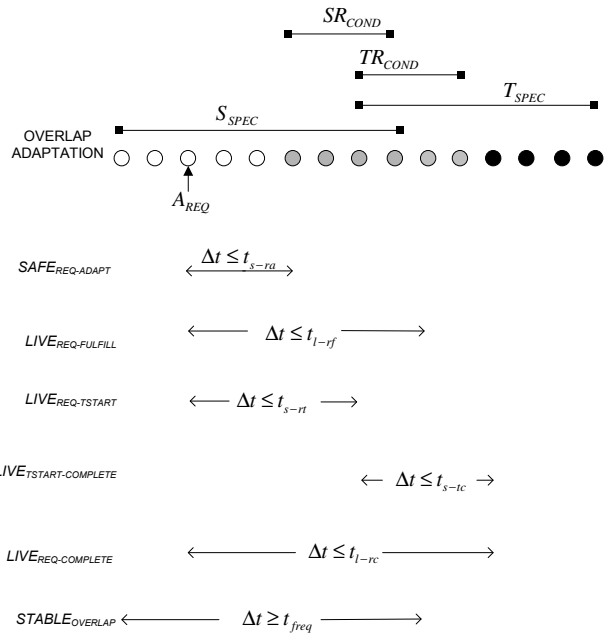


Fig. 5 Real-time constraints for overlap adaptation.

6 Case Study

To illustrate the use of TA-LTL in specifying adaptation timing properties, we use TA-LTL to specify the safety, liveness, and stability of the adaptation behavior of the adaptive audio streaming system mentioned earlier. The interconnection of the systems is depicted in Figure 6. A live audio stream is multicast from a wired desktop computer to multiple mobile devices via the 802.11b WLAN. Effectively, the receivers are used as multicast-capable Internet "phones" participating in a conferencing application. We expect the system to operate in environments with different packet loss rates. The loss rate of the wireless connection changes over time and the program should adapt its behavior accordingly: When the loss rate is low, the sender/receiver should use a low loss-tolerance and low bandwidth-consuming encoder/decoder or should remove the encoder/decoder completely; when the loss rate is high, they should use a high-loss-tolerance and consequently high bandwidth-consuming encoder/decoder.

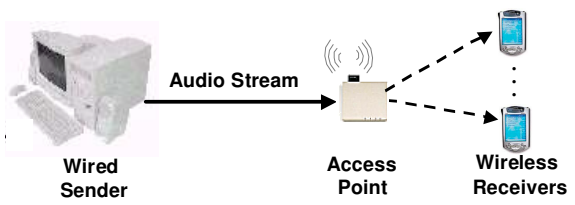


Fig. 6 Audio streaming system connection

6.1 Forward Error Correction (FEC) Filters

The adaptive behavior in this system is realized using MetaSockets [32] that internally support dynamic reconfiguration of a chain of processing filters. MetaSockets can be used in place of regular Java sockets, providing the same imperative functionality, including methods for sending and receiving data. However, their internal structure and behavior can be adapted at run time in response to changes in their environment. MetaSockets behavior can be adapted through the insertion and removal of *filters* that manipulate the data stream. For example, filters can perform encryption, decryption, forward error correction, compression, and so forth. Specifically, in this case study, we use two different FEC filters to respond to the dynamic changing loss rate. One filter uses block erasure codes [27], and the other uses the GSM 06.10 encoding algorithm also used for cellular telephones [12]. The FEC encoded data is different from the original audio data and can be played only after it is decoded by the FEC decoder. A run-time scenario is to have all collected audio samples at the sender side stored in a data buffer waiting to be encoded by the FEC encoder. The encoded data packets are then multicast to the receivers and stored in a data buffer waiting to be decoded the FEC decoder. The decoded audio data will eventually be played by the receivers. Here, we use the guided adaptation semantics to specify the adaptation of inserting an FEC filter at the receiver side.

The use of (n, k) *block erasure codes* for error correction was popularized by Rizzo [27] and is now used in many wired and wireless distributed systems. Figure 7 depicts the basic operation of these codes. An encoder converts k source packets into n encoded packets, such that any k of the n encoded packets can be used to reconstruct the k source packets [27]. In this paper, we use only *systematic* codes, which means that the first k of the n encoded packets are identical to the k source packets. We refer to the first k packets as *data* packets, and the remaining $n - k$ packets as *parity* packets. Each set of n encoded packets is referred to as a *group*. The advantage of using block erasure codes for multicasting is that any x $(x \le n - k)$ parity packet can be used to correct independent x packet losses among different receivers [27]. In the remainder of the paper, we refer to the block-oriented FEC simply as "FEC (n, k)". While block-oriented FEC approaches are effective in improving the quality of interactive audio streams on wireless networks, the group sizes must be relatively small in order to reduce playback delays.

An alternative approach with lower delay and lower overhead is the *GSM-oriented FEC encoding* (also known as *signal processing-based FEC (SFEC)*) [8], in which a lossy, compressed encoding of each packet p_i is piggybacked onto one or more subsequent packets. If packet p_i is lost, but one of the encodings of packet p_i arrives at the receiver, then at least a lower quality version of the

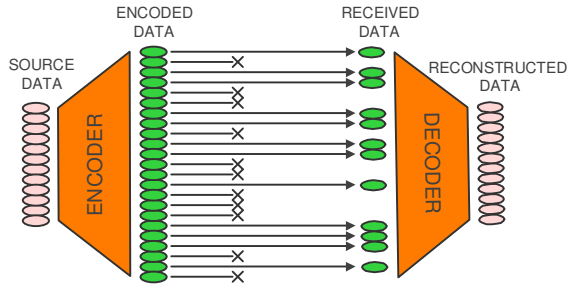
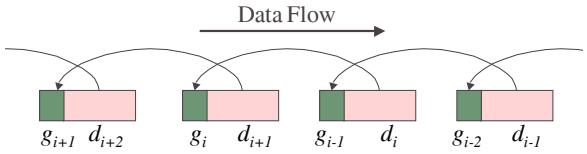
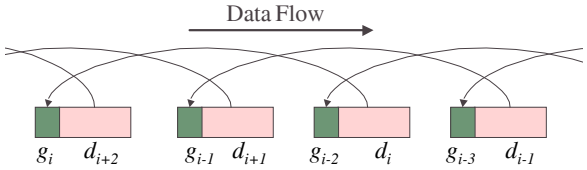


Fig. 7 Operation of block erasure code.

packet can be played to the listener. The parameter θ is the offset between the original packet and its compressed version. Figure 8 shows two different examples, one with $\theta = 1$ and the other with $\theta = 2$. It is also possible to place multiple encodings of the same packet in multiple subsequent packets, such as using both $\theta_1 = 1$ and $\theta_2 = 3$. Although GSM is a CPU-intensive coding algorithm [8], the bandwidth overhead is very small. In the remainder of the paper, we refer to the GSM-oriented FEC simply as "GSM (θ, c)," which means copies of the coded packet p are placed in c successive packets, beginning from the θ^{th} packets after p.



(a) GSM encoding with $\theta = 1$



(b) GSM encoding with $\theta = 2$

Fig. 8 Different ways of using GSM encoding on a packet stream (d_i: data, g_i: copy).

6.2 Specifying QoS Constraint with TA-LTL

In previous research [40], we have investigated the quality-of-service (QoS) properties (such as packet delivery rate and delay) of these two FEC protocols. Table 1 shows the loss rate perceived by the receiving application, that is, *after FEC decoding*, for different FEC (n, k) or GSM (θ, c) settings. Another factor important to real-time communication is the additional delay introduced into the packet stream by the encoding and decoding filters. Table 2 shows the worst case delay introduced by different FEC codes to wait for the encoded packets. For example, considering FEC (8, 4) and GSM (3, 1), if the first data packet is lost, then the receiver will need to wait for at least 3 packets until the first parity packet or piggybacked packet arrives to recover the loss.

From the QoS comparison we know that for a certain wireless environment (e.g., 28% raw loss rate as shown in Table 1), different FEC codes have different loss recovery performance while holding different timing characteristics. For example, if the raw loss rate of the wireless environment changes to 28% and the expected loss rate after FEC decoding is $\leq 10\%$, FEC (8, 4), GSM (1, 2), GSM (1, 3), GSM (2, 3), GSM (2, 3), GSM (3, 2), and GSM (3, 3) are possible filter candidates for adaptation. However, in order to satisfy the real-time interactive audio streaming requirement, the accumulated delay after adaptation should not exceed 150 msec [28]. For this example, in the guided adaptation, the restriction condition is that the data is blocked from transmission during the adaptation. Thus the accumulated delay comprises two parts: the adaptation delay and the FEC delay. If there is no additional operation (e.g., dropping packet, fast playback) to reduce the delay, this accumulated delay will remain in the post-adaptation communication. Thus we can use TA-LTL to specify the real-time constraint, and the selection of FEC filters and the implementation of adaptation should satisfy this constraint.

This adaptation semantics and its timing properties can be specified by A-LTL and TA-LTL as shown in Figure 9. In Formula 15, $SPEC_{FEC}$ states that initially the system is running without FEC filters (the source program), i.e., it satisfies $NoFEC_{SPEC}$. After an adaptation request, A_{REQ}, is received, the system should satisfy a restriction condition R_{COND}, which requires the receiver to temporarily stop receiving incoming encoded FEC data. When the source program reaches a quiescent state, i.e., all original audio data in the buffer have been played, the system stops satisfying the $NoFEC_{SPEC}$, and starts to satisfy $WithFEC_{SPEC}$. In Formula 16, $SAFE_{FEC}$ states that upon receiving the FEC decoder insertion adaptation request, the source program should stop receiving incoming packets within t_{safe}, otherwise the encoded FEC data will be received and stored in the data buffer, causing a failure since it cannot be played directly. In Formula 17, $LIVE_{FEC}$ specifies the real-time constraint of the adaptation for inserting FEC filters discussed above. Specifically, t_{live} is the criteria for ensuring real-time audio communication (150 msec in this study). In Formula 18, $STABLE_{FEC}$ prevents the system from inserting/removing the FEC filter too frequently. This example illustrates the use of TA-LTL in specifying quanti-

Table 1 Loss rate comparison of different FEC codes

Code	Raw	GSM (1,1)	GSM (1,2)	GSM (1,3)	GSM (2,1)	GSM (2,2)	GSM (2,3)	GSM (3,1)	GSM (3,2)	GSM (3,3)	FEC (4,4)	FEC (6,4)	FEC (8,4)
%	28	11.05	6.28	3.53	13.88	6.8	3.78	10.27	4.8	2.75	28.20	16.29	9.16

Table 2 Delay comparison of different FEC codes

Code	GSM (1,1)	GSM (1,2)	GSM (1,3)	GSM (2,1)	GSM (2,2)	GSM (2,3)	GSM (3,1)	GSM (3,2)	GSM (3,3)	FEC (4,4)	FEC (6,4)	FEC (8,4)
Delay (msec)	19.95	39.95	59.9	40.83	60.37	80.10	62.55	82.52	102.87	17.79	37.73	52.33

tative timing requirements for adaptive communication software.

$$\begin{aligned} SPEC_{FEC} = {} & NoFEC_{SPEC} \wedge (\Diamond A_{REQ} \\ & \xrightarrow{\Omega_1} R_{COND}) \\ & \xrightarrow{\Omega_2} WithFEC_{SPEC} \end{aligned} \tag{15}$$

$$\begin{aligned} SAFE_{FEC} = {} & \Diamond x.(A_{REQ} \\ & \xrightarrow{\Omega_1} y.R_{COND} \wedge (x + t_{safe} \geq y)) \\ & \xrightarrow{\Omega_2} WithFEC_{SPEC} \end{aligned} \tag{16}$$

$$\begin{aligned} LIVE_{FEC} = {} & \Diamond x.(A_{REQ} \\ & \xrightarrow{\Omega_1} R_{COND} \\ & \xrightarrow{\Omega_2} y.WithFEC_{SPEC} \wedge (y \leq x + t_{live})) \end{aligned} \tag{17}$$

$$\begin{aligned} STABLE_{FEC} = {} & x.(NoFEC_{SPEC} \wedge (\Diamond A_{REQ} \\ & \xrightarrow{\Omega_1} R_{COND}) \\ & \xrightarrow{\Omega_2} y.WithFEC_{SPEC} \wedge (y \geq x + t_{freq})) \end{aligned} \tag{18}$$

Fig. 9 Specifying adaptation semantics with TA-LTL

7 Discussion

In this section, we discuss issues regarding TA-LTL, including its expressiveness, decidability, model checking algorithm, and asynchronous extensions.

7.1 Expressiveness

We say a logic L_1 is *more expressive* than another logic L_2, denoted $L_1 \geq L_2$, if and only if for any formula ϕ_2 in L_2, there exists a formula ϕ_1 in L_1 such that ϕ_1 accepts exactly the same set of *models* that ϕ_2 accepts. We say a logic L_1 is *strictly more expressive* than another logic L_2, denoted $L_1 > L_2$, if and only if $L_1 \geq L_2$, but $L_2 \ngeq L_1$. We say L_1 and L_2 are equivalent in expressive power, denoted $L_1 \equiv L_2$, if and only if $L_1 \geq L_2$ and $L_2 \geq L_1$. The expressiveness of TA-LTL is defined in terms of the set of timed state sequences that satisfy each TA-LTL formula.

The TA-LTL introduced in this paper can be considered as a combination of A-LTL and TPTL. A-LTL is a non-real time temporal logic, and thus it does not reference τ, the time sequence, in a timed state sequence. Therefore, TA-LTL is strictly more expressive than A-LTL.

Theorem 1: *TA-LTL is strictly more expressive than A-LTL.*

Proof: We prove that TA-LTL is more expressive than A-LTL but not vice-versa.

(1) TA-LTL is more expressive than A-LTL: TA-LTL $\geq$ A-LTL.

The proof of this statement is straightforward. Since TA-LTL is a super set of A-LTL, for any formula ϕ of A-LTL, there is also a formula ϕ' of exactly the same form in TA-LTL, and $\phi' = \phi$. Therefore, we have TA-LTL $\geq$ A-LTL

(2) A-LTL is not more expressive than TA-LTL: A-LTL $\ngeq$ TA-LTL.

For the timed state sequence

$$\sigma = (\neg p, 0), (p, 1), (p, 2), \cdots \text{ and}$$
$$\sigma' = (\neg p, 0), (p, 5), (p, 6), \cdots,$$

there is a formula in TA-LTL, $\phi = \Diamond x.p \wedge (x < 5)$, that distinguishes these two sequences. That is, $\sigma \models_{inf} \phi$, while $\sigma' \not\models_{inf} \phi$. However, for any formula ψ in A-LTL, we have $\sigma \models \psi$ if and only if $\sigma' \models \psi$. Therefore, there is no formula in A-LTL that accepts exactly the same set of sequences as ϕ does. Thus we have A-LTL $\not\geq$ TA-LTL.

Therefore, TA-LTL $>$ A-LTL

■

We have previously proved that A-LTL and LTL are equivalent in their expressive power, whereas, A-LTL is at least exponentially more succinct than LTL in expressing adaptation semantics [36]. We can conclude that TA-LTL is at least exponentially more succinct than TPTL, however, the relative expressive power between TA-LTL and TPTL is yet to be determined, which is part of our ongoing work.

7.2 Decision Procedure and Model Checking

For a given logic, a formula ϕ of the logic is *satisfiable* if and only if there exists a model σ, such that $\sigma \models \phi$. A *decision procedure* of a logic determines whether each formula of the given logic is satisfiable. Tableaux-based decision procedures have been adopted by researchers to address the satisfiability problem of temporal logics including LTL [22], the choppy logic [29], etc. Previously we also proposed using a tableaux-based decision procedure for the satisfiability problem for A-LTL [37]. Alur *et al.* [3] extended the traditional tableaux-based approach with time-difference propositions to address the real-time properties in TPTL satisfiability problem. The same technique can be used to extend the A-LTL decision procedure for TA-LTL.

Model checking techniques determine whether a program (formally a Kripke structure [13]) satisfies a given temporal logic formula by exploring the state space of the program. We adopt the timed state graph [3] as the formal model for real-time programs. A timed state graph is a tuple $T = \langle L, \mu, \nu, L_0, \mathfrak{T} \rangle$, where

- L is a finite set of locations;
- $\mu : L \rightarrow 2^P$ is a labeling function that labels each location $l \in L$ with a state $\mu_l \subseteq P$;
- ν is a time difference function that labels each location with the time different from its predecessor location;
- $L_0 \subseteq L$ is a set of initial locations;
- $\mathfrak{T} \subseteq L^2$ is a set of transitions.

A computation of the timed state graph is a timed state sequence $\rho = (\sigma, \tau)$ if and only if there is an infinite path $l = l_0, l_1, \cdots$ of T, such that $\sigma_i = \mu_{l_i}$, and the time difference in $l_i(+1)$ equals $(\tau_{i+1} - \tau_i)$. The model checking of a timed state graph T against a TA-LTL formula ϕ proceeds as follows:

1. Calculate the negation of the formula $\neg\phi$.
2. Constructe the tableau $T(\neg\phi)$ for $\neg\phi$.
3. Calculate the product $P = T * T(\neg\phi)$
4. Determining whether the produce P contains an accepting path σ starting from a state that contains the initial state of T and $\neg\phi$.

The program model T satisfies the formula ϕ if and only if there does not exist such a path σ.

7.3 Asynchronous Multi-Clock Extension

TA-LTL is a formalism for specifying real-time properties with a single system clock. However, in a distributed system, multiple local clocks may be used among different system components, where a *local clock* is a physical device that when accessed, returns a clock reading. As a result, it may not be possible to compare the numerical readings of these clocks to determine the temporal precedence, since they may tick at completely independent rates. To interpret timing inequalities involving different local clocks, Wang *et al* proposed APTL [33], an asynchronous distributed system model, to treat local clock readings not as numerical values, but as special temporal marks, enabling comparison of the "temporal precedence" between readings from different local clocks. For a system of m local clocks, a_i is the reading a of clock i. Given a local clock reading a_i and an integer constant d, $a_i + d$ is a *displaced local clock reading* with *displacement* d and defined as $a_i + d \equiv (a + d)_i$.

In order to specify adaptation semantics for distributed systems, we could also extend TA-LTL with an asynchronous clock system as follows:

$$\pi := x_i + c \mid c \tag{19}$$

$$\phi := p \mid \pi_1 \sim \pi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \stackrel{\Omega}{\frown} \phi_2 \mid [x_1, \ldots, x_m]\phi \tag{20}$$

for $x \in V$, $p \in P$, and integer constants $c \in N$.

The local clocks are indexed with positive integers 1 through m. The variables $x_1, \ldots, x_m$ are names of local reading variables for clock $1, \ldots, m$, respectively. The variable i is the integer index for a clock. The timing constraints ($\pi_1 \sim \pi_2$) are the forms of *asynchronous inequalities* proposed in APTL which presents the temporal relationship between local clock readings [33]. Corresponding to the usual arithmetic inequality operators $<, \leq, >, \geq, =$, there are five *asynchronous inequality operators*:

$\lhd$: earlier than;
$\trianglelefteq$: no later than;
$\rhd$: later than;
$\trianglerighteq$: no earlier than;
$\rightleftharpoons$: at the same instant as.

In this paper, our discussion focuses on specifying adaptation semantics with timing constraints in the context of a single clock. We assume that the request thread, the source program, and the target program share the same clock. Adaptation semantics with multiple clocks can be considered a generalization of the semantics discussed in this paper and is part of our ongoing work.

8 Conclusions

Adaptation semantics should be precisely specified at the requirements level so that they can be well understood and correctly implemented in later phases of the software development process. This paper introduced the specification of adaptation timing properties in autonomic systems in terms of TA-LTL.

After an adaptation temporal specification is constructed, it may serve as guidance for the adaptation developers to clarify the intent for the adaptive program. It can also be used to check for consistency in the temporal logic specifications. By using run-time verification techniques, we may also automatically identify the safe states for an adaptation and insert adaptation logic at appropriate points in the program. We may even perform model checking to verify the correctness of the program model against the temporal logic specifications.

Our future work includes investigating adaptation semantics with multiple clocks, automatically generating program models and code based on a given adaptation semantics, developing model checking tools to verify the correctness of adaptive programs [36].

Acknowledgements. This work has been supported in part by NSF grants EIA-0000433, EIA-0130724, ITR-0313142, CCR-9901017, CCF-0541131, and CNS-0551622, and the Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and by Siemens Corporate Research, and a Michigan State University Quality Fund Concept Grant.

Further Information. A number of related papers and technical reports of the Software Engineering and Network Systems Laboratory can be found at the following URL: `http://www.cse.msu.edu/sens`.

References

1. RAPIDware. http://www.cse.msu.edu/rapidware/, Software Engineering and Network Systems Laboratory, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan
2. Allen R, Douence R, Garlan D (1998) Specifying and analyzing dynamic software architectures. In: Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98), Lisbon, Portugal
3. Alur R, Henzinger TA (1994) A really temporal logic. Journal of the ACM 41(1):181–203
4. Appavoo J, Hui K, Soules CAN, Wisniewski RW, Silva DMD, Krieger O, Auslander MA, Edelsohn DJ, Gamsa B, Ganger GR, McKenney P, Ostrowski M, Rosenburg B, Stumm M, Xenidis J (2003) Enabling autonomic behavior in systems software with hot swapping. IBM Systems Journal, Special Issue on Autonomic Computing 42(1)
5. Bellini P, Mattolini R, Nesi P (2000) Temporal logics for real-time system specification. ACM Computing Surveys 32(1):12–42
6. Berry DM, Cheng BHC, Zhang J (2005) The four levels of requirements engineering for and in dynamic adaptive systems. In: Proceedings of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality, Porto, Portugal
7. Bershad BN, et al (1994) Spin - an extensible microkernel for application-specific operating system services. Tech. rep., Dept. of Computer Science and Engineering, University of Washington
8. Bolot JC, Vega-Garcia A (1996) Control mechanisms for packet audio in Internet. In: Proceedings of IEEE INFOCOM'96, San Francisco, California, pp 232–239
9. Bowman H, Thompson SJ (1998) A tableaux method for Interval Temporal Logic with projection. In: TABLEAUX'98, International Conference on Analytic Tableaux and Related Methods, Springer-Verlag, no. 1397 in Lecture Notes in AI, pp 108–123, URL `http://www.cs.ukc.ac.uk/pubs/1998/528`
10. Chen WK, Hiltunen MA, Schlichting R (2001) Constructing adaptive software in distributed systems. In: Proceedings of the 21st International Conference on Distributed Computing Systems, Phoenix, AZ, pp 635–643, URL `http://citeseer.nj.nec.com/chen01constructing.html`
11. Cuesta CE, de la Fuente P, Barrio-Solarzano M (2001) Dynamic coordination architecture through the use of reflection. In: Proceedings of the 2001 ACM Symposium on Applied Computing, ACM Press, New York, NY, USA, pp 134–140
12. Degener J, Bormann C (2000) The GSM 06.10 lossy speech compression library and its applications. Available at http://kbs.cs.tu-berlin.de/ jutta/toast.html

13. Emerson EA (1990) Temporal and modal logic. Handbook of theoretical computer science (vol B): formal models and semantics pp 995–1072
14. Jahanian F, Mok AK (1986) Safety analysis of timing properties in real-time systems. IEEE Transaction of Software Engineering 12(9):890–904
15. Kephart JO, Chess DM (2003) The vision of autonomic computing. IEEE Computer 36(1):41–50
16. Kon F, Román M, Liu P, Mao J, Yamane T, Magalhães LC, Campbell RH (2000) Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer-Verlag, New York, no. 1795 in LNCS, pp 121–143
17. Koymans R (1990) Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4):255–299
18. Kramer J, Magee J (1990) The evolving philosophers problem: Dynamic change management. IEEE Transactions on Software Engineering 16(11):1293–1306
19. Kramer J, Magee J, Sloman M (1992) Configuring distributed systems. In: Proceedings of the 5th Workshop on ACM SIGOPS European, ACM Press, New York, NY, USA, pp 1–5
20. Kulkarni SS, Biyani KN (2004) Correctness of component-based adaptation. In: Proceedings of the 7th International Symposium on Component-Based Software Engineering, pp 48–58
21. Kulkarni SS, Biyani KN, Arumugam U (2003) Composing distributed fault-tolerance components. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), Supplemental Volume, Workshop on Principles of Dependable Systems, pp W127–W136
22. Lichtenstein O, Pnueli A (1985) Checking that finite state concurrent programs satisfy their linear specification. In: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, pp 97–107, DOI http://doi.acm.org/10.1145/318593.318622
23. McKinley P, Sadjadi SM, Kasten EP, Cheng BHC (2004) Composing adaptive software. IEEE Computer 37(7):56–64
24. Ostroff J, Wonham W (1987) Modeling and verifying real-time embedded computer systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp 124–132
25. Pal PP, Loyall J, Schantz RE, Zinky JA, Webber F (2000) Open implementation toolkit for building survivable applications. In: Proceedings of the DARPA Information Survivability Conference and Exposition, Hilton Head Island
26. Pnueli A (1977) The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium Foundations of Computer Science (FOCS 1977), pp 46–57
27. Rizzo L (1997) Effective erasure codes for reliable computer communication protocols. ACM Computer Communication Review
28. Rosenberg J, Qiu L, Schulzrinne H (2000) Integrating packet FEC into adaptive voice playout buffer algorithms on the Internet. In: Proceedings of IEEE INFOCOM 2000, pp 1705–1714
29. Rosner R, Pnueli A (1986) A choppy logic. In: Proceedings of Symposium on Logic in Computer Science, Cambridge, Massachusetts, USA, pp 306–313
30. Sadjadi S, McKinley P, Kasten E, Zhou Z (2006) Metasockets: Design and operation of run-time reconfigurable communication services. Software Practice and Experience 36:1157–1178
31. Sadjadi SM, McKinley PK (2004) ACT: An adaptive CORBA template to support unanticipated adaptation. In: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS), Tokyo, Japan
32. Sadjadi SM, McKinley PK, Kasten EP (2003) Architecture and operation of an adaptable communication substrate. In: Proceedings of the Ninth IEEE International Workshop on Future Trends in Distributed Computing, San Juan, Puerto Rico
33. Wang F, Mok A, Emerson EA (1992) Formal specification of asynchronous distributed real-time systems by APTL. In: Proceedings of the 14th International Conference on Software Engineering, ACM Press, New York, NY, USA, pp 188–198
34. Zhang J, Cheng BHC (2005) Specifying adaptation semantics. In: Proceedings of the 2005 Workshop on Architecting Dependable Systems, ACM Press, New York, NY, USA, pp 1–7
35. Zhang J, Cheng BHC (2006) Model-based development of dynamically adaptive software. In: Proceedings of International Conference on Software Engineering (ICSE'06), Shanghai,China
36. Zhang J, Cheng BHC (2006) Modular model checking of dynamically adaptive programs. Tech. Rep. MSU-CSE-06-18, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, URL `http://www.cse.msu.edu/~zhangji9/Zhang06Modular.pdf`, `http://www.cse.msu.edu/~zhangji9/Zhang06Modular.pdf`
37. Zhang J, Cheng BHC (2006) Using temporal logic to specify adaptive program semantics. Journal of Systems and Software (JSS), Architecting Dependable Systems 79(10):1361–1369
38. Zhang J, Yang Z, Cheng BHC, McKinley PK (2004) Adding safeness to dynamic adaptation techniques. In: Proceedings of ICSE 2004 Workshop on Architecting Dependable Systems, Edinburgh, Scotland, UK
39. Zhang J, Cheng BHC, Yang Z, McKinley PK (2005) Enabling safe dynamic component-based software adaptation. Architecting Dependable Systems, Lecture Notes in Computer Science pp 194–211
40. Zhou Z, McKinley PK, Sadjadi SM (2004) On quality-of-service and energy consumption tradeoffs in FEC-enabled audio streaming. In: Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQoS 2004), Montreal, Canada
41. Zhou Z, Zhang J, McKinley PK, Cheng BHC (2006) TA-LTL: Specifying adaptation timing properties in autonomic systems. In: 3rd IEEE Workshop on Engineering of Autonomic Systems (EASe 2006), Columbia, Maryland

Ji Zhang received his BE degree (1999) and his ME degree (2001) in computer science and engineering from Tsinghua University, Beijing, China. He is currently a PhD student in the Department of Computer Science and Engineering at Michigan State University, East Lansing, Michigan. His current research interests are software development process, formal methods, model-checking, model-based software development, testing, autonomic computing, software architecture, aspect-oriented programming, and algorithms. He is a member of the ACM SIGSOFT and a member of the IEEE. He can be contacted electronically at zhangji9@cse.msu.edu.

Zhinan Zhou received Ph.D. degree (2006) in computer science and engineering at Michigan State University. He also received his B.E. degree (1999) in ther-

mal engineering and his M.E. degree (2001) in computer science and engineering from Tsinghua University, Beijing, China. He currently works as a software engineer in the San Jose Mobile Communications Lab of Samsung Telecommmunications America. His research interests include Pervasive Mobile Computing, Wireless Mobile Application, Wireless Multimedia Communications and Services, Adaptive Middleware, Autonomic Computing, and System Integration. He is a member of IEEE. He can be reached at z.zhou@samsung.com.

Betty H.C. Cheng received her BS in computer science from Northwestern University in 1985 and her MS and PhD degrees in computer science from the University of Illinois-Urbana Champaign, in 1987 and 1990, respectively. She is currently a professor in the Department of Computer Science and Engineering at Michigan State University. Her interests include formal and automated methods for software engineering, model-driven development, specification and modeling patterns, and dynamically adaptive software, all in the context of high assurance systems. Industrial partners include Siemens, Motorola, and General Dynamics. Her research has been funded by NSF, ONR, NASA, EPA, and numerous industrial organizations. She serves on the editorial boards for IEEE Transactions on Software Engineering, Requirements Engineering, and Software and Systems Modeling, as well as numerous program and organizational committees for international conferences and workshops. She may be contacted electronically at chengb@cse.msu.edu.

Philip K. McKinley received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign. Dr. McKinley is currently a Professor in the Department of Computer Science at Michigan State University, where he has been on the faculty since 1990. He was previously a member of technical staff at Bell Laboratories. Dr. McKinley has served as an Associate Editor for IEEE Transactions on Parallel and Distributed Systems and was co-chair of the program committee for the 2003 IEEE International Conference on Distributed Computing Systems. His current research interests include self-adaptive software, mobile computing, group communication, and digital evolution. He is a member of the IEEE and ACM. He may be contacted electronically at mckinley@cse.msu.edu.