Online Appendix to: MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems

DIDAC GIL DE LA IGLESIA, Linnaeus University
DANNY WEYNS, Linnaeus University

A. TIMED AUTOMATA AND TIMED COMPUTATIONAL TREE LOGIC

A timed automaton (TA) is a finite-state machine extended with clock variables that models a behavior. Clock variables are used to synchronize behaviors. Additionally, automata can communicate through channels, where the sender behavior *x!* synchronizes with the receiver behavior $x?$.

A timed automaton is a tuple $(N, l_0, T, Label, C, clock, guard, invariant)$ [Bengtsson and Yi 2004] in which:

N	a non-empty, finite set of locations (or nodes) with an initial location $l_0 \in N$;
$T \subseteq L \times L$	a set of transitions;
$Label : N \to 2^{AP}$	a function that assigns to each location $l \in N$ a set $Label(l)$ of atomic positions;
C	a finite set of clocks;
$clock : T \to 2^C$	a function that assigns to each transition $t \in T$ a set of clocks clocks(t);
$guard : T \to \Psi(C)$	a function that labels each transition $t \in T$ with a clock constraint $guard(t)$ over C;
$inv : N \to \Psi(C)$	a function that assigns to each node an invariant.

In this work, we use UPPAAL [Behrmann et al. 2006] to model TA. In UPPAAL behavior specifications can be complemented with expressions specified in a C-like language to define data structures (struct concept) and functions. We follow UPPAAL symbol's conventions for the description of the behavior templates, presented in Table IV.

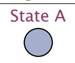
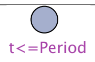
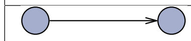
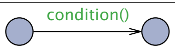
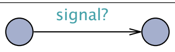
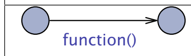
The following points apply to the specification of the behaviors and should be considered to understand internal node behaviors and interactions between the automata.

— Automata do not synchronize except with *signal?* / *signal!*
— Conditions are evaluated locally by the node executing a given automaton
— Functions are executed locally on the node executing a given automaton
— Conditions are evaluated before a transition is selected for execution
— Functions are executed (synchronously) as a result of selecting a given transition
— A transition may have no function/signal associated with it

Transitions between TA's states can fire either by an event-triggering or time-triggering. With event-triggering one automaton triggers another via a signal sent through a channel. This case is illustrated in Fig. 24-left, where an automaton with a behavior *B1* fires a *signal* in order to trigger a transition on the behavior *B2*. In this case, we say that the second behavior *B2* is dependent on *B1*, as this would not be able to execute transitions without the corresponding signal from the first. Optionally, data may be transferred from a behavior *B1* to a *B2* using a knowledge repository. With time-triggering a transition is fired based on state invariants and time conditions. This case is shown

© 2015 ACM 1556-4665/2015/01-ARTX $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

Table IV. Conventions in Timed Automata figures

Figure	Name	Description
	State	States of behaviors are represented by circles and (optionally) annotated in red on top of the associated *state*.
	Committed State	Committed states are represented with circles containing the *c* character. Committed states must be left without time consumption.
	Urgent State	Urgent states of behaviors are represented with circles containing the *u* character. Urgent states must be left as soon as exiting conditions are found (normally defined by conditions on outgoing transitions).
	Initial State	Initial states of behaviors are represented by double-lined circles. There must be one unique *Initial State* per automaton, specifying the behavior state when the system starts.
	Invariants	Invariants that need to be satisfied in certain states are annotated in purple under the related *state*.
	Transition	Transitions between two states are represented by directional arrows, showing the origin and destination of the transition.
	Conditions	Conditions determine the possibility for a transition to be taken. Conditions to enable firing of transitions between states are annotated in green under the related *transition*.
	Signal	Signals used for communication between behaviors are annotated dark blue over the associated *transition*.
	Function	Functions associated with behaviors are annotated in light blue under the associated *transition*. Functions are executed when the related transition is taken.

in Fig. 24-right, where a behavior *B2* autonomously executes transitions in the automaton on a time-based approach (*Tick*). Due to the autonomy of the behavior with respect to another *B1* behavior, time triggering requires a data repository to store shared knowledge whenever information needs to be transferred between behaviors. In general, time-triggering is less efficient in terms of verification (as it implies more execution threads).

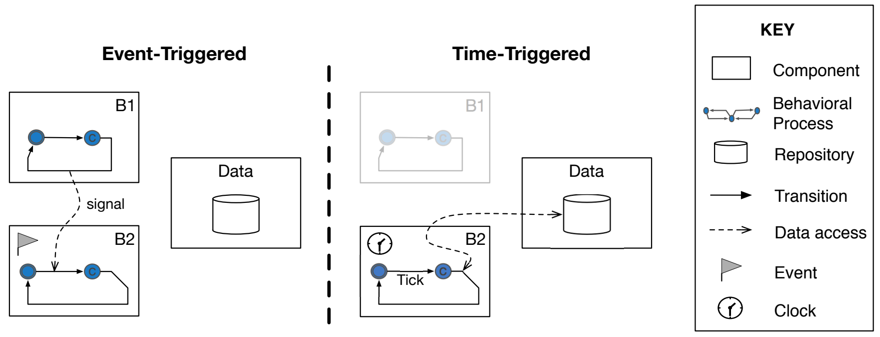


Fig. 24. Event- (left) and time- (right) triggering approaches for component behaviors

Fig. 25 and Fig. 26 show examples of event triggering and time triggering approaches respectively for a Monitor behavior in a GPS based scenario. In the event triggered approach (Fig. 25), behavior leaves the *Monitoring* state when triggered by an event (represented by the *GPSwentDown[PiD]?* signal). In this example, the *GPSwentDown[Pid]!* event informs about a required GPS quality threshold being unsatisfied.

Once triggered, the automaton performs specific actions designed for the monitor behavior and comes back to the *Monitoring* state. These actions include the collection of the most recent information regarding the monitored resource (which is identified and labelled with an ID, *caseID=getCase()*), the information regarding the group in which this resource is being used (*myMVD=determineMyMVD(Pid)*) and an update of the Knowledge(*remove_Phone(myMVD)*). An identification of the case is required in order to avoid redundant events informing about the same GPS quality problem.

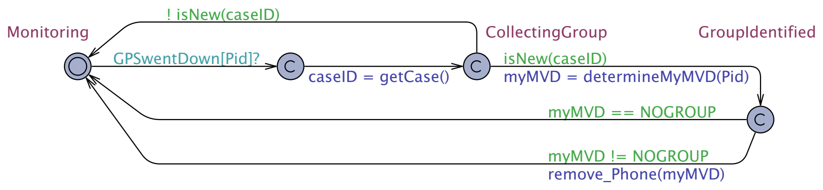


Fig. 25. Formal specification behavior with event triggering approach

In the time triggered approach (Fig. 26), a transition in the behavior is triggered periodically via a clock (*t*) and a defined period (5 seconds in the example). On this defined period basis, the automaton leaves the *Monitoring* state in order to perform the specific monitor actions. Before coming back to the *Monitoring* state, the automaton requires that the clock is initialized (*t=0*) in order to allow further time based triggers. In the time triggered approach, as the behavior is triggered on a period basis instead of detected changes in the quality of the GPS module, it is important to check whether errors have been found or not (*caseID==NONE*) before further actions are performed.

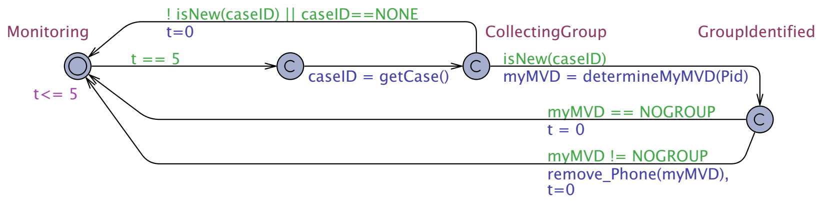


Fig. 26. Formal specification behavior with time triggering approach

UPPAAL supports the design of property specification using timed-computational tree logic (TCTL). TCTL is a formal language for property specification based on TCL extended with clock variables. The syntax supported by UPPAAL for property specification is defined as follow [Henzinger et al. 1994]:

$$\phi ::= p \mid !\phi \mid \phi \, \vee \, \phi \mid EX \, \phi \mid E[\phi \, \cup \, \phi] \mid A[\phi \, \cup \, \phi] \mid z.\phi$$

Where:

ϕ	is a property to be specified;
p	is an atomic proposition or a clock constraint;
EX	is an expression applied on a property;
E	expresses the existence of a path that fulfills a property;
A	expresses the invariant fulfillment of a property;
z	expresses a state predicate

Below, we explain the different symbols that can be used in the TCTL sintaxis.

$E[]p$	There is a *path* in which p will *always hold*. For example, there is a path where a GPS module has always NULL coordinates. This is a broken during manufacturing GPS module.
$E <> p$	It is *possible to reach* a state in which p is satisfied. For example, a GPS module will eventually acquire position coordinates.
$A[]p$	p *holds invariantly*. For example, location coordinates are always on earth.
$A <> p$	p is inevitable. It will *eventually happen*. For example, a GPS will eventually fail in gathering accurate positions.
$A - - > B$	*If* A becomes true, then B will *inevitably be true*. For example, if a GPS module turns on, it will eventually collect GPS coordinates.
$A \; imply \; B$	If A becomes true, B will become *true at the same time*. For example, if a GPS acquires location coordinates, then some accuracy values with respect to the location coordinates are obtained.