# Towards Automated and Correct Composition of Timed Services

Daniel Stöhr and Sabine Glesner

Technical University of Berlin, Chair Software Engineering for Embedded Systems,
{daniel.stoehr,sabine.glesner}@tu-berlin.de
www.pes.tu-berlin.de

**Abstract.** The design of programs controlling distributed components in safety-critical domains can be very error-prone and time-consuming. Especially in the presence of real-time requirements the correctness with respect to functional and non-functional properties must be guaranteed. To this end, we develop a technique for the automated and correct composition of timed services based on timed i/o automata, the temporal logic TCTL, and planning algorithms based on model checking. Thus, we can speed up the development of controller programs while assuring correctness.

## 1 Introduction

The development of controller programs coordinating distributed components in safety and time-critical environments is a very complex task. On the one hand, the correctness with respect to functional and non-functional properties, like timed behaviour, has to be assured. This can be achieved by using verification techniques, e.g., model checking, which automatically proves total correctness but significantly increases the quantity and length of development cycles. On the other hand, development time has to be short to achieve a small time-to-market. In our work, we address the problem of closing the gap between these opposites with development methods assuring correctness by construction.

For this purpose, we lift the problem of creating a controller model to the problem of automated service composition. Our method shall be able to generate controller models for a given set of timed services with respect to functional and non-functional (especially timed) requirements. We describe the behaviour of the services as *timed i/o automata* (*TIOA*) and the composition requirements in the temporal logic *Timed CTL* (*TCTL*). The generated controller, which is an orchestrator in the context of service-oriented computing, is a TIOA handling the input and output actions of the individual automata. To realize our approach, we extend *planning as model checking* which is already able to deal with *untimed* automata-based planning domains and CTL goals. We extend the theory of planning as model checking by introducing time in terms of real-valued clocks. In this paper, we report on our first results with which we can automatically generate plans for a set of timed services and requirements given in a restricted subset of TCTL.

A particularly interesting domain for our approach is the synchronization of medical devices. Here, functional and real-time requirements have to be met in

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 319–331, 2013.

order to guarantee the patient's safety. We show the applicability of our approach through case study where distinct components of a PCA (patient-controlled analgesia) pump have to be controlled to guarantee functional and time-related requirements. In our case study, we have closed components within a device that can be controlled over a central software. Likewise, other scenarios where distinct devices need synchronization over a network are also possible.

Briefly summarized, the contributions of this paper are:

- We outline an approach realizing the automated composition of timed services by utilizing planning algorithms based on model checking. Timed i/o automata describe the input services, TCTL formulas describe the composition requirements.
- We present our extensions of the existing planning algorithms and we have implemented a prototype realizing planning for a subset of TCTL.
- We provide a case study for our approach that cannot be handled by current composition and planning techniques.

The rest of this paper is structured as follows. In Section 2 we give the background for our work. We present the concept of *timed i/o automata* and *TCTL*, along with a running case example for our approach. Also, we discuss *planning as model checking*. In Section 3 we describe the concept of our overall approach, and present our extensions to the symbolic planning algorithms. Afterwards, in Section 4, we describe our prototype implementation. In Section 5 we discuss related works for automated service composition and temporal planning. Finally, in Section 6, we give a conclusion and outline future work.

## 2 Preliminaries

In this section, we present the background of our work. In Section 2.1, we describe *timed i/o automata* and in Section 2.2, the temporal logic *TCTL*. In both sections, we introduce the case example for our approach. Finally, in Section 2.3 we describe the concept of *planning as model checking* in detail.

### 2.1 Timed I/O Automata (TIOA) and Case Example

*Timed i/o automata* (*TIOA*), as proposed by David and Larsen [1], enrich the concept of timed automata by adding input and output signals and the concept of parallel composition over signals belonging together.

A TIOA is a tuple $A = (Loc, q_0, Clk, E, Act, Inv)$ where $Loc$ is a finite set of locations, $q_0 \in Loc$ is the initial location, and $Clk$ is a finite set of clocks. $E \subseteq Loc \times Act \times \mathcal{B}(Clk) \times 2^{Clk} \times Loc$ is a set of edges with timed guards ($\mathcal{B}(Clk)$ are all Boolean expressions over $Clk$) and a set of clocks to be reset. Finally, $Act = Act_i \cup Act_o \cup \{\tau\}$ is a finite set of actions consisting of disjunct sets of input and output actions and the symbol $\tau$ for transitions without actions. $Inv : Loc \mapsto \mathcal{B}(Clk)$ is a set of location invariants.

The *parallel composition* $TA_1 || TA_2$ of TIOA is similar to building up the crossproduct. If an in- and output signal belong together, both transitions are combined as one parallel transition.

It has been shown that web service compositions, e.g. in *WS-BPEL* or *WS-CDL*, can be translated into timed automata and vice versa [2]. Service instances are represented as single automata, messages as synchronized actions, and wait durations or timeouts as guards and invariants. Thus, web services compositions can be analyzed and verified by model checking.

Figure 1 shows the visual representation of a TIOA and gives a use case for our approach. It shows a simplified version of a system controlling a PCA pump [3]. These pumps administer the pain medication for a patient by delivering drug doses at a basal rate. The complete use case contains further requirements. E.g., the pump can raise alarms or the patient can signal the system to give a bolus injection if his pain level is too high. However, the first-mentioned requirement is sufficient for illustrating the benefit our approach, as it cannot be solved by existing tools for automated composition or AI planning (see Section 5).

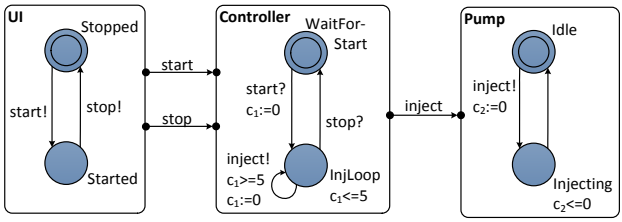


**Fig. 1.** Composition of TIOA describing the components of a PCA pump and a controller

The system consists of a TIOA `UI` representing the user interface that turns the pump on and off via the output signals `start` and `stop`. The other component is described by `Pump` which just waits for the input signal `inject` triggering an injection dose and resetting the clock $c_2$. Then, the state `Injecting` is entered and left without letting time progress. This is realized by using an invariant.

The TIOA `Controller` performs the composition according to the above-mentioned requirement. With our approach, we generate such a controller out of a formal requirements specification. In its initial State `WaitForStart`, the controller waits for the signal `start` to turn on the system and to reset the clock $c_1$. The drug injection has to be triggered in a specific interval (here, 5 time units). The invariant of `InjLoop` and the guard and clock reset on the loop transition ensure that the signal `inject` is emmited every 5 time units. Thus, the controller guarantees the injection rate defined in the composition requirements[1].

[1] Our composition requirements leave some flexibility in the controller's design. The controller shown in our example triggers the first injection 5 time units after the system has been started. A controller injecting immediately after the start would also be valid but would have to guarantee that the user cannot bypass the safetey requirement by switching fast between `start` and `stop`. Our extended planning algorithms would generate the latter version. However, to improve the comprehensibility of our example we have chosen the former version.

### 2.2 Timed CTL (TCTL)

The *Timed Computation Tree Logic* (*TCTL* [4]) is an extension of CTL with clock constraints. TCTL can be used to describe conditions on branching paths in timed i/o automata. The language is defined as follows.

$$\Phi := p \mid (x + c) \leq (y + d) \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathbf{E}(\Phi_1 \mathbf{U} \Phi_2) \mid \mathbf{A}(\Phi_1 \mathbf{U} \Phi_2) \mid x.\Phi$$

where $p$ is an atomic proposition, $x$ and $y$ are clock variables, and $c, d \in \mathbb{N}$.

The semantics of the logical ($\wedge$, $\neg$) and relational ($\leq$) operators are defined as usual. The temporal operator $\mathbf{E}(\Phi_1 \mathbf{U} \Phi_2)$ holds iff there exists a path preserving $\Phi_1$ until $\Phi_2$ holds. $\mathbf{A}(\Phi_1 \mathbf{U} \Phi_2)$ is defined analogously for all paths. $x.\Phi$ is the reset quantifier. Iff clock $x$ is zero, then $\Phi$ holds.

Logical ($\vee$, $\rightarrow$, $\leftrightarrow$) and relational ($<$, $\geq$, $>$) abbreviations are defined as usual, too. The temporal abbreviations $\mathbf{EF}p$ and $\mathbf{EG}p$ hold iff there exists a path where $p$ holds eventually, respectively forever. $\mathbf{AF}p$ and $\mathbf{AG}p$ are defined analogously for all paths. Finally, $\mathbf{EF}_{<c}p$ with $c \in \mathbb{N}$ holds iff there exists a path where eventually $p$ holds in less than $c$ time units.

In the following, we show the requirements of our case study (see Figure 1) expressed as TCTL formulas. In our approach, these formulas are used to automatically generate the controller.

(1) $\mathbf{AG}(\texttt{Started} \rightarrow (\mathbf{AG}\ \mathbf{AF}\texttt{Injecting}))$
(2) $\mathbf{AG}(\texttt{Injecting} \rightarrow \neg\mathbf{EF}_{<5}\texttt{Injecting})$
(3) $\mathbf{AG}(\texttt{Injecting} \rightarrow \texttt{Started})$

Requirement *(1)* describes that whenever the system is started, the drug injection must take place repeatedly. Requirement *(2)* defines the injection rate. Whenever the state `Injecting` has been entered, it may not be reached again in less than 5 time units. In this scenario, our extended planning tool would solve the goal $\mathbf{AF}\texttt{Injecting}$ as soon as possible. Hence, these two requirements are sufficient for generating a controller injecting every 5 time units. At last, we need our safety requirement *(3)*, stating that the injection may only be triggered while the system is activated.

### 2.3 Planning as Model Checking

Methods of AI planning generate plans solving problems in a given domain. In *planning as model checking* [5] the domain is described in an automata-based model and the planning problem consists of sets of initial and goal states. A plan is a list stating which action has to be performed in which state, to reach a goal state from an initial state.

However, the problem of identifiying paths leading to the goal is similar to the counter example generation in model checking where a path leading to a violated property is computed. Hence, planning as model checking reduces the planning problem to a model checking problem. As described in Section 3 we extend planning as model checking to cope with time and realize the automated composition approach.

In the following, we give the basic definitions for planning domains, planning problems, and plans. Afterwards, we show how the planning problem can be solved via symbolic model checking.

**Basic Definitions.** A *planning domain* is a tuple $D = (P, S, A, R)$ where $P$ is a finite set of propositions, $S \subseteq 2^P$ is a finite set of states, $A$ is a finite set of actions, and $R \subseteq S \times A \times S$ is the transition relation.

A *planning problem* is defined as a tuple $P = (D, I, G)$ where $D$ is a planning domain, $I \subseteq S$ is a set of initial states, and $G \subseteq S$ is a set of goal states.

Plans for a given domain $D = (P, S, A, R)$ are expressed as a *state-action table* $\pi \subseteq \{(s, a) | s \in S, a \in A, \exists s' \in S : R(s, a, s')\}$. State-action tables can be *executed* within a domain by starting in an initial state and performing the corresponding action for each state. With regard to non-determinism, the execution may take different *execution paths* through the domain.

Finally, a *strong plan* for a domain $D = (P, S, A, R)$ and problem $P = (D, I, G)$ is a state-action table where each execution path starting in a state of $I$ leads to a state in $G$. A strong plan is calculated by using the following function:

$Exec(s, a) = \{s' | R(s, a, s')\}$
$StrongPreImage(S) = \{(s, a) | \emptyset \neq Exec(s, a) \subseteq S\}$

$StrongPreImage$ explores all state-action pairs that lead into the set of states $S$ while taking non-determinism into account. The planning algorithm applies that function on the set of goal states, then, on the union of the newly explored and goal states, and so on. The plan generation fails if a fixpoint is reached before all initial states are explored. Otherwise, the set of all collected state-action pairs forming the strong plan, is returned[2].

Planning with CTL formulas as planning goal is also offered [5]. Here, the complex goal is divided into smaller subgoals that are resolved by using functions similar to the preimage function.

**Symbolic Planning.** Planning as model checking uses techniques of symbolic model checking to solve the planning problem with logical operations. At first, the planning domain is described by Boolean formulas. Then, the preimage function can also be described symbolically by using logical operators.

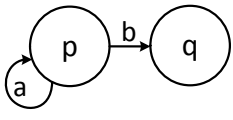

**Fig. 2.** A planning domain

We give a small example for a planning domain in Figure 2, consisting of two states $p$ and $q$, a loop transition on $p$ using the action $a$, and a transition between $p$ and $q$ using action $b$. The domain is described by the following fomulas:

$T_1 \doteq p \wedge a \wedge p'$ $\quad$ $States \doteq (p \wedge \neg q) \vee (\neg p \wedge q)$
$T_2 \doteq p \wedge b \wedge q'$ $\quad$ $Actions \doteq (a \wedge \neg b) \vee (\neg a \wedge b)$
$T \doteq T_1 \vee T_2$ $\quad$ $D \doteq T \wedge States \wedge Actions$

Every state of the domain is represented by two Booleans $p$, $p'$, $q$, and $q'$. Unprimed state variables describe the source of a transition, the others the destination. The actions are represented by the boolean variables $a$ and $b$.

[2] The complete algorithm uses an additional function ensuring that a state is not explored twice. We have ommited that function due to lack of space.

$T_1$ describes the loop transition where each conjunct describes an element of the tuple describing the transition in the set-based representation. $T_2$ describes the other transition. The set of Transitions is described by $T$. The formulas *States* and *Actions* state that only one state or action may be active at once. Finally, $D$ represents the whole planning domain.

Building on the system's symbolic description, the preimage function is defined symbolically, too. To this end, *quantified Boolean formulas* (QBFs) are used. QBFs extend propositional logics by universal and existential quantifiers. If $\Phi$ is a formula and $v$ one of its variables, then $\forall v.\Phi$ is equivalent to $\Phi[v/True] \wedge \Phi[v/False]$. The existential quantification is defined analogously using the logical or.

The symbolic preimage function is defined as follows, where S is the formula describing the set of input states (e.g., $p' \vee q'$ or just $p'$) and $D$ is the formula describing the planning domain.

$$StrongPreImage(S) \doteq \forall p', q'.(D \rightarrow S) \wedge \exists p', q'.D$$

The preimage function is applied in the same way described for the set-based function. Other operations performed in the complete algorithm, are also performed using logical operators. By doing this, very large sets of states can be handled in compact formulas and, hence, the amount of memory usage and computation time can be reduced.

## 3 Automated Composition of Timed Services

In this section, we describe our approach for the automated and correct composition of timed services. A set of service interfaces, expressed as TIOA, and composition requirements, expressed as TCTL formulas, serve as input. The output is a generated orchestrator, i.e. a central automaton. We do so by using planning algorithms based on model checking where the set of services becomes the planning domain and the composition requirements become planning goals.

Firstly, in Section 3.1 we outline the overall concept of our approach and how we transform the composition problem into a planning problem. Secondly, in Section 3.2, we give a detailed description of our extensions to *planning as model checking*, which realize the composition process. At this moment, we have realized the extensions for a subset of TCTL (formulas in the form of $\mathbf{A}(\neg\phi\mathbf{U}\psi)$ describing unsafe states $\phi$ and a goal state $\psi$). However, our results can be extended for full TCTL support.

### 3.1 Methodology of Our Approach

With our approach, we transform the problem of automatically composing timed services into an AI planning problem. The methodology is visualized in Figure 3. As input we take a set of TIOA $TA_1, ..., TA_n$ describing service interfaces. $TA_{||}$, the parallel composition of these automata, serves as

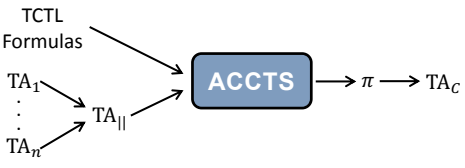


**Fig. 3.** Our approach for the Automated and Correct Composition of Timed Services (ACCTS)

planning domain. As composition requirements, which describe the properties of the overall composition, we use TCTL formulas reasoning on states and timing constraints. In our example the input automata are `UI` and `Pump`, while the composition requirements are a conjunction of the functional and safety requirements.

Afterwards, we use our algorithms for timed planning as model checking to build up a plan $\pi$ controlling the domain with respect to the composition requirements. The plan is used to build up our controller $TA_C$, the TIOA that orchestrates the overall composition.

Existing planning techniques cannot deal with timed domains and requirements of that expressiveness (see Section 5.2). Hence, we decided to extend the algorithms of *planning as model checking* that already solves CTL goals in planning domains based on untimed automata. To overcome this restriction, we introduce clocks into the planning theory. In the next section, we describe first extensions we have realized.

### 3.2 Extending Strong Planning with Time

In the following, we describe how we cope with time by extending the structures and algorithms of planning as model checking (see Section 2.3). Analogously to TIOA (see Section 2.1), we add clock variables, guards, and invariants. For additionally extending the expressiveness of strong goals, we also introduce a possibility to express unsafe states, which a plan has to avoid. Thus, we can plan with TCTL formulas in the form of $\mathbf{A}(\neg\phi\mathbf{U}\psi)$ where $\phi$ describes the set of unsafe states and $\psi$ describes the set of goal states. Subsequent, we describe how we shifted our extensions to the symbolic representation.

**Basic Definitions.** A *timed planning domain* is a tuple $D_T = (P, S, Clk, A, R, Inv)$. As before, $P$ is a finite set of propositions, $S \subseteq 2^P$ is a finite set of states, and $A$ is a finite set of actions. Additionally, $Clk$ is the set of clock variables and $R \subseteq S \times A \times B(Clk) \times 2^{Clk} \times S$ is the transition relation enriched with a guard and a set of clocks to be reset. $Inv : S \mapsto \mathcal{B}(Clk)$ are the timed invariants of the states.

A *timed planning problem* is defined as a tuple $P_T = (D_T, I, U, G)$ where $D_T$ is a timed planning domain and $I$, $U$, $G \subseteq S \times \mathbb{R}_{\geq 0}^{Clk}$ are initial, unsafe, and goal states connected to clock values. Here, $\mathbb{R}_{\geq 0}^{Clk}$ (an abbreviation for the mapping $Clk \mapsto \mathbb{R}_{\geq 0}$) is a *valuation* of the clock variables.

For $(s, v), (s', v') \in S \times \mathbb{R}_{\geq 0}^{Clk}$ (a state and corresponding clock valuation) a discrete transition $(s, v) \xrightarrow{a} (s', v')$ exists, if there exists a tuple $(s, a, g, res, s') \in R$ such that $v' = v[res]$, $v \models g$, and $v' \models Inv(s')$. $v[res]$ denotes the valuation $v$ with all clock in $res$ reset to zero.

We express plans for a given timed domain $D_T$ as a *timed state-action table* $\pi_T \subseteq \{((s, v), a | (s, v) \in S \times \mathbb{R}_{\geq 0}^{Clk}, a \in A, \exists (s', v') \in S \times \mathbb{R}_{\geq 0}^{Clk} : (s, v) \xrightarrow{a} (s', v')\}$. The *execution* of a plan and the resulting *execution paths* do not only consist of performing actions. Also *time progress*, called time transitions, can occur where

the clock valuation is increased for all clocks synchronously (within the bounds of the invariants).

Finally, a *strong plan* for a timed domain $D_T$ and timed problem $P$ is a state-action table where each execution path starting in a state of $I$ leads to a state in $G$. For the calculation of strong plans, we extend the preimage function, as follows:

$$Exec((s,v),a) = \{s' | (s,v) \xrightarrow{a} (s',v')\}$$
$$S \swarrow = \{(s, v-d) | (s,v) \in S \wedge d \in \mathbb{R}_{\geq 0} \wedge v - d \geq 0 \wedge (s, v-d) \models Inv(s)\}$$
$$StrongPreImage(S,U) =$$
$$\{((s,v),a) | \emptyset \neq Exec((s,v),a) \subseteq S \swarrow \wedge v \models Inv(s) \wedge s \notin U\}$$

We extended the *Exec* function which computes all possible results of executing an action in a state and which now considers guards and invariants of $s'$ by using discrete transitions. Also, we define the $\swarrow$ operator which is common in the model checking of timed automata. The argument is a set of states $S$ and the result are all states that can be reached from $S$ by letting time progress backwards. In our extended *StrongPreImage* function, we apply the $\swarrow$ operator to the set of states $S$ (which are the goal states and already discovered states leading to the goal). Thus, we ensure that not only the execution of actions but also time progress is considered. The preimage function now calculates those states that can reach a state in $S$ immediately by executing an action, or by letting time progress and then executing an action. Moreover, we now enforce that the *StrongPreImage* function considers the invariant of the state $s$ and ensure that $s$ is not part of the unsafe states in $U$.

In the fixpoint algorithm, the *StrongPreImage* function is initially applied to the goal states, then to the states of the resulting state-action pairs, and so on. Thus, we explore the states that can reach the goal states while avoiding unsafe states. The algorithm terminates if the initial states are reached and fails if a fixpoint is reached. Our extended algorithm still terminates, because we have a finite set of states, clock values cannot be lower than 0, and we only use the $\swarrow$ operator. At some point of time, the algorithm will have explored all states and will have counted all clocks to 0 (resp. to the lowest bound enforced by the invariants). Thus we will reach a fixpoint from any given set of states and corresponding clock valuations.

**Mapping TIOA into Timed Planning Domains.** Our extensions allow us to map the service interfaces, resp. TIOAs, into timed planning domains. As described in Section 3.1, we do so by building the parallel composition of all required services. The resulting single automaton $A_{||} = (Loc, q0, Clk_A, E, Act, Inv_A)$ can easily be mapped into a timed planning domain $D_T = (P, S, Clk_D, A, R, Inv_D)$.

Firstly, we have a property in $P$ for each location in *Loc*. The set of domain states $S$ ensures that only one property may be true at a time[3]. The sets of clock

[3] For higher efficiency it is possible to encode the set of locations as a combination of property values. Another possibility is to allow more than one active property at a time by translating the states of the original parallel automata. However, investigating the efficiency impact of these alternative solutions will be part of future work.

variables $Clk_A$ and $Clk_D$, as well as the action labels $Act$ and $A$ are identical. Now the transition relation $R$ and the invariants $Inv_D$ can be built up out of $E$ and $Inv_A$ by matching properties and corresponding locations.

**Symbolic Planning.** For the symbolic planning algorithms, we firstly enrich the system's symbolic representation clock variables and clock constraints. We illustrate this, by using the `Pump` automaton (see Section 2.1) transformed into a timed planning domain:

$T_1 \doteq Idle \wedge inject! \wedge c_2' = 0 \wedge Injecting'$
$T_2 \doteq Injecting \wedge \neg inject! \wedge c_2' = c_2 \wedge Idle'$
$T \doteq T_1 \vee T_2$
$States \doteq (Idle \wedge \neg Injecting) \vee (\neg Idle \wedge Injecting \wedge c_2 \leq 0)$
$Actions \doteq (inject! \vee \neg inject!)$
$D \doteq T \wedge States \wedge Actions$

As in the untimed version, we still have variables for all propositions and actions. We add the real-valued clock variables $c_1$, $c_1'$, $c_2$, and $c_2'$ for describing clock values before or after discrete transitions, and for describing invariants. Formula $T_1$ describes the left transition of `Pump`. Source, action, and destination are denoted as before. We add the expression $c_2' = 0$ describing the clock reset. If the transition contained a guard, the correpsonding exepression would refer to $c_2$.

Also, we need to consider clock variables in the *States* formula. Here, we add the invariant $c_2 \leq 0$ of $Injecting$. The other formulas, $T_2$, $T$, *Actions*, and $D$ are built up as before, since they do not directly refer to clock variables. The expression $\neg inject!$ means that no action is active and stands for $\tau$ in the set-based representation.

For the symbolic representation of our extended preimage function, we introduce further quantifications over the clock variables, the $\swarrow$ operator, and consider the unsafe states $U$:

$$StrongPreImage(S, U) \doteq \forall p', q', c_1', c_2'.(D \rightarrow S\swarrow) \wedge \exists p', q', c_1', c_2'.(D \wedge \neg U)$$

The additional quantifications remove the clock resets from the formula $D$, leaving only the preimage states, corresponding actions, and guards. The result characterizes the states that can reach $S$ immediately or after time progress.

In summary, our approach provides a methodology for the automated composition of timed services. By transforming the composition problem into a planning problem and by using techniques of model checking, we ensure the correctness of resulting compositions. Within our approach, service interfaces are represented as TIOA and composition requirements as TCTL formulas. So far, we have realized extensions to the existing planning algorithms, that allow us to solve planning goals in the form of $\mathbf{A}(\neg\phi\mathbf{U}\psi)$ where $\phi$ are the unsafe states and $\psi$ are the goal states. However, our results can be further extended for full TCTL support. In Section 6 we outline how these extensions can be realized.

## 4 Implementation with MDD+CRDs

For evaluating our first extensions, we have implemented a prototype. To work with logical formulas and operators, we used the open source library *REDLIB* [6]

which offers *multi-decision- & clock-restriction-diagrams* (*MDD+CRDs*), a data structure representing Boolean formulas with Boolean, discrete, and clock variables. The usual logical operators, variable quantification, and the $\swarrow$ operator are also included. MDD+CRDs were developed for the model checking of timed automata and RED, a model checker based on REDLIB, performs very well in time and space compared to other established model checking tools [7]. Algorithms of model checking generally lead to a high complexity. But by using REDLIB, we hope to improve the scalability of our implementation.

Our prototype takes an xml representation of TIOA and a planning problem, resp. a set of goal and unsafe states, as input. The output is a plan solving the planning problem. Our whole algorithm is executed by performing logical operations.

We tested our prototype by generating a plan for our case study with the simplified, non-cyclic composition requirement $\mathbf{A}(\neg\texttt{Injecting}\ \mathbf{U}\ (\texttt{Started} \wedge \texttt{Injecting}))$. Also, we generated plans for other small examples. Our evaluation led to promising results where plans have been generated within milliseconds (under Ubuntu 10.04 on an Intel i5, 2,7 Ghz with 2GB RAM). However, for a profound evaluation of the scalability, we will need much larger examples and full TCTL support.

## 5 Related Work

In this section, we present works related to our approach. In Section 5.1, we discuss approaches for automated service composition. In Section 5.2, we examine techniques for temporal planning.

### 5.1 Automated Service Composition

The majority of techniques for automated service composition do not consider time as a service property or composition requirement. The *astro* framework [8], e.g., uses planning as model checking to automatically generate a BPEL composition out of a given set of web services and composition requirements. Here, the way how the planning theory was utilized to solve the composition problem is similar to our approach. This work is not able to deal with timed service behaviour or timed requirements. However, timed capabilities are one of the main characteristics of our proposed approach.

Most approaches on the composition of timed services regard time as a measurement for communication latency between world-wide distributed services [9] or telephone servers [10]. They consider time as a *Quality of Service* criterion that serves as parameter for choosing a proper service instance during the composition process. In our work, we see time as a part of the service's behaviour and functionality itself.

To the best of our knowledge, there is only one approach realizing automated service composition with timed composition requirements [11]. In contrast to our approach, this work offers a very low degree of automation because the overall workflow of a BPEL composition has to exist before timed requirements can

be woven into the workflow according to specification. In contrast to that, our approach offers a very high degree of automation by generating the workflow of the composition from scratch.

We proposed our approach in an earlyier work [12]. This work only presented the overall concept of the approach. Now, we have refined the concept itself, realized first theoretical foundations, and implemented a prototype.

### 5.2 Temporal Planning

There are several temporal planning tools of which most only allow planning goals that consist of one or more goal states and deadlines [13,14]. Others create a schedule for a list of atomic actions with regard to specified interaction constraints [15]. These tools are not applicable to our domain because we need to support more complex TCTL goals. In the following, we discuss two temporal planning tools that are interesting with regard to the expressiveness of their goal languages.

The TALPlanner [16] plans over domains and goals expressed in a first order logic, the temporal action logic (TAL). TAL formulas allow the quantification over discrete or clock variables and provide operators that put actions and variable changes in a causal correlation. In contrast to TCTL, TAL specifies linear (opposed to branching) behavior. Moreover, TAL reasons on actions actions, while TCTL reasons on states. Hence, TAL does not cover the expressiveness of TCTL.

Uppaal-TIGA [17] solves game-theoretical problems on timed game automata with respect to reachability or safety properties expressed in a subset of TCTL. These goals do not allow the nesting of TCTL operators or the conjunction of goals using temporal operators. Hence, it is not possible to consider, e.g., the cyclic appearing reachability goal we use in our example (requirement (1) in Section 2.2). The formulas that can be expressed are hard-coded into the algorithms. In constrast to that, the results of our apporach can be continued for full TCTL support.

## 6 Conclusion and Future Work

In this paper, we have presented first theoretical foundations of our approach for the automated and correct composition of timed services by using timed i/o automata, TCTL, and planning algorithms based on model checking. With our first extensions to the existing untimed planning algorithms, we can generate plans fulfulling composition requirements expressed in a subset of TCTL (formulas in the form of $\mathbf{A}(\neg\phi\mathbf{U}\psi)$). Thus, we already cover those composition requirements where a state has to be reached under occlusion of unsafe states. Moreover, we have implemented a first prototype realizing our extended algorithms. We evaluated our approach by generating the plan for a simplified version of our case study where the composition requirements can be expressed in the above-mentioned subset of TCTL.

In future work, we will realize the support of whole TCTL formulas as planning goals, similar to the extensions for strong planning. Yet, no other automated composition and no planning approach is able to deal with with the expressiveness of full TCTL formulas. The existing algorithms for simple CTL goals [5] divide complex goal formulas into atomic subgoals. According to the corresponding CTL operators, the reachability of subgoals is symbolically calculated by using functions similar to the preimage function. For supporting clock variables, we will extend these functions in a similar way, as we did here.

Furthermore, we introduce TCTL's reset quantifier, $x.\Phi$, as a new subgoal. $x.\Phi$ bounds a new clock variable $x$ to an expression $\Phi$. In the planning context, $\Phi$ is a set of subgoals. To solve these goals, we add the new clock variable to the system's symbolic description and assure that $x$ must be zero every time a subgoal of $\Phi$ is activated.

Morover, we want to realize the translation of the generated plans into an orchestrator, i.e., a central timed i/o automaton performing the composition. Hence, we will adapt and extend a work translating plans into BPEL compositions without considering time [8]. Moreover, we will perform a larger case study where a PET/CT scanner has to be synchronized with other medical devices over a network. This allows us to investigate the scalability of our approach in the context of real-life applications.

## References

1. David, A., Larsen, K.G., Legay, A., Nyman, U.: Timed I/O Automata: A Complete Specification Theory for Real-time Systems. Science (2010)
2. Gregorio, D., Cambronero, M.E., Pardo, J.J., Cuartero, F.: Automatic generation of Correct Web Services Choreographies and Orchestrations with Model Checking Techniques. In: International Conference on Internet and Web Applications and Services, AICT-ICIW 2006 (2006)
3. Arney, D., Jetley, R., Jones, P., Lee, I., Sokolsky, O.: Formal Methods Based Development of a PCA Infusion Pump Reference Model: Generic Infusion Pump (GIP) Project. In: Joint Workshop on High Confidence Medical Devices, Software and Systems and Medical Device Plug-and-Play Interoperability (2007)
4. Alur, R.: Techniques for Automatic Verification of Real-Time Systems. PhD thesis, Stanford University (1991)
5. Pistore, M., Bettin, R., Traverso, P.: Symbolic techniques for planning with extended goals in non-deterministic domains. In: 6th European Conference on Planning (2001)
6. Wang, F.: REDLIB A Library of Integrated BDD-like Diagrams for Dense-Time Model Verification (2012)
7. Wang, F.: Efficient verification of timed automata with BDD-like data structures. International Journal on Software Tools for Technology Transfer 6(1), 77–97 (2004)
8. Pistore, M., Traverso, P., Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. Artificial Intelligence 174 (2005)
9. Moussa, H., Gao, T., Yen, I.L., Bastani, F., Jeng, J.J.: Toward effective service composition for real-time SOA-based systems. Service Oriented Computing and Applications 4(1) (2010)

10. Lin, L., Lin, P.: Orchestration in Web Services and Real-Time Communications. IEEE Communications Magazine (July 2007)
11. Kallel, S., Dinkelaker, T., Mezini, M., Charfi, A., Jmaiel, M.: Specifying and Monitoring Temporal Properties in Web services Compositions. In: Seventh IEEE European Conference on Web Services (2009)
12. Stöhr, D., Glesner, S.: Automated Composition of Timed Services by Planning as Model Checking. In: Proceedings of the 4th Central European Workshop on Services and their Composition (2012)
13. Do, M.B., Kambhampati, S.: Sapa: A Domain-Independent Heuristic Metric Temporal Planner. In: Proceedings of European Conference on Planning (2001)
14. Garrido, A., Fox, M., Long, D.: A Temporal Planning System for Durative Actions of PDDL2.1. In: European Conference on AI, vol. 1 (2002)
15. Muscettola, N.: HSTS: Integrating Planning and Scheduling. Intelligent Scheduling (1993)
16. Doherty, P., Kvarnström, J., Heintz, F.: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. In: Autonomous Agents and Multi-Agent Systems, vol. 19(3) (2009)
17. Cassez, F., David, A., Fleury, E., Larsen, K.G.: Efficient On-the-fly Algorithms for the Analysis of Timed Games (2005)