Abstract—Current Web service composition approaches and languages such as WS-BPEL do not allow to define temporal constraints in a declarative and separate way. Also it is not possible to verify if there are contradictions between the temporal constraints implemented in the composition. These limitations lead to maintainability and correctness problems. In this paper, we tackle these problems through a novel approach to temporal constraints in Web service compositions, which combines formal methods and aspect-oriented programming. In this approach, we use a powerful and expressive formal language, called XTUS-Automata, for specifying time-related properties and we introduce specification patterns that ease the definition of such constraints. The formal specifications are translated automatically into AO4BPEL aspects, which ensure the runtime monitoring of the temporal constraints. Our approach enables a declarative, separate, and verifiable specification of temporal properties and it generates automatically modular enforcement code for those properties.

Keywords—Aspect-oriented programming, Runtime monitoring, Temporal Properties, Formal methods, Web services

I. INTRODUCTION

Web services composition is being used in various domains, such as business to business integration (B2B) or enterprise application integration (EAI). Current composition approaches focus on the specification of a set of interactions between Web services together with the flow of control and data around them and often neglect other important concerns such as temporal properties or quality of service.

Temporal constraints are essential in several Web service composition scenarios as time plays a crucial role in business processes and B2B interactions. Current Web service composition approaches and languages, such as the standard WS-BPEL [1], provide language constructs for managing time. E.g., WS-BPEL provides special activities such as onAlarm or wait that can be used to implement temporal constraints. However, the WS-BPEL approach has several limitations: (1) to express temporal constraints, one needs to program them manually in the process specification using WS-BPEL activities; (2) only simple time expressions such as timeouts and durations may be expressed; (3) the correctness of a Web service composition after the manual insertion of time related activities cannot be verified; (4) the process code implementing the temporal constraints is mixed and tangled with the process logic.

To recap, current Web service composition approaches do not allow a declarative and separate specification of temporal constraints. This leads to maintainability problems as the designer of the composition cannot modify the temporal constraints without understanding the whole process logic and identifying the process activities implementing a given temporal constraint. Further, Changes to temporal constraints cannot be done at runtime, i.e., one has to stop the composite Web service, undeploy the corresponding process, change the process, and redeploy it. In addition to the maintainability problem, there is a correctness problem that arises when not using a formal language. For instance, when temporal properties are implemented using BPEL activities one cannot verify if there are inconsistencies or deadlock states because of lacking formal verification tools.

In this paper, we address the limitations explained above through a novel approach to temporal constraints in Web service compositions, which combines formal methods and aspect-oriented programming [2]. Our approach covers both the specification of temporal constraints using a new formal language and their monitoring through the automatic generation of AO4BPEL [3] aspects.

The contributions of this work are many-fold: First, we present a new formal language, called XTUS-Automata that combines timed automata (TA) [4] and extended time unit system (XTUS) [5]. This language allows specifying temporal properties involving relative time and absolute time. Second, we present generic specification patterns that ease the specification of temporal properties in Web services compositions. Third, we present an aspect-based monitoring mechanism, in which formally specified temporal constraints are translated automatically to modular aspect code in the aspect-oriented workflow language AO4BPEL.
As AO4BPEL supports dynamic weaving the aspects can be integrated at runtime with the processes. The generated aspects intercept the execution of process activities. If the respective temporal properties are satisfied, the activity will be executed; otherwise, the aspect prohibits the activity execution and throws an exception. Figure 1 gives an overview of our approach which consists of three steps.

In the first step, implementation, the designer defines the Web service composition using WS-BPEL but without implementing any temporal properties.

In the second step, formal specification, the designer specifies formally the temporal properties using XTUS-Automata, which is a combination of Timed Automata with the eXtended Time Unit System. Thereby, the designer should follow and respect so-called specification patterns, which we provide for different types of temporal properties. In addition, the activities names and the parameters used in the formal specification step should be the same as those used in the WS-BPEL process. After this step, formal verification of deadlock freedom and other properties can be done using existing model checking tools such as UPPAAL [6].

In the third step, aspect-based runtime monitoring, the designer uses our aspect generator tool to automatically generate monitoring code consisting of AO4BPEL aspects that monitor the temporal constraints at runtime and ensure that they are satisfied. If that is the case, the process activities are executed; otherwise, the aspect prohibits the execution of the monitored activities and throws an exception.

The remainder of this paper is organized as follows. Section II gives background and introduces a running example. Section III presents the formal language and introduces patterns for formal specification of temporal properties in Web service composition. Section IV describes the mapping of the formal specifications to aspect code. Section V reports on related works and Section VI concludes the paper.

II. BACKGROUND AND EXAMPLE

For the sake of a better understanding of our extensions, we review in this section, the concepts of timed automata, extended time unit system, and the AO4BPEL language. In addition, we introduce the example of a travel agency scenario to explain the problems of current composition approaches and the envisaged solution.

A. Timed Automata

Timed Automata (TA) [4] specify the behavior of real-time systems over time. They extend finite-state automata with time constraints using continuous clock variables. Clocks can be reset at certain transitions and their values can be used in constraints for enabling/disabling transitions.

Definition1: Timed Automata

A Timed Automaton \( A \) is a 6-tuple \((L, l_0, E, C, Inv, T)\):
- \( L \) is a finite set of locations,
- \( l_0 \in L \) is an initial location,
- \( E \) is a set of events/actions,
- \( C \) is a finite set of clocks,
- \( Inv \) is a mapping that labels each location \( l \in L \) with some inequalities of the form \( c \leq n \), for some clock \( c \) and integer \( n \),
- \( T : L \times E \times 2^C \times \Phi(C) \times L \) is the transition relation. \((l, e, \lambda, \varphi, l')\) is a transition from location \( l \) to location \( l' \) on event \( e \). \( \lambda \) gives the clocks to be reset with this transition, and \( \varphi \) is a clock constraint over \( C \) that specifies when the switch is enabled. The set of clock constraints \( \Phi(C) \) is defined by the following grammar:

\[
\varphi ::= c_1 \sim n | c_1 - c_2 \sim n | \neg \varphi | \varphi \land \varphi
\]
where \( \sim \in \{<, \leq, \geq\} \), \( c_1, c_2 \in C \), and \( n \in \mathbb{N} \).

A transition \((l, e, \lambda, \varphi, l')\) \( \in T \) is enabled if the control is at location \( l \), the clock constraint \( \varphi \) is satisfied and the event \( e \) is enabled. After taking the transition, the control moves to location \( l' \) and the clocks in \( \lambda \) are reset.

Clock constraints are evaluated over clock valuations. The clock valuation \( v : C \rightarrow \mathbb{R}_{\geq 0} \) is a function from the set of clocks to the non negative reals \( r \). A valuation \( v \) satisfies the clock constraint \( \varphi \), denoted by \( v \models \varphi \), if

\[
\begin{align*}
v \models c < r \iff & v(c) < r \\
v \models c \leq r \iff & v(c) \leq r \\
v \models \neg \varphi \iff & v \not\models \varphi \\
v \models \varphi_1 \land \varphi_2 \iff & v \models \varphi_1 \text{ and } v \models \varphi_2
\end{align*}
\]
Definition 2: Semantics of Timed Automata
The semantics of a timed automaton $A (L, l_0, E, C, Inv, T)$ is defined as a labeled transition system $(S, s_0, \rightarrow)$, where $S \subseteq L \times \mathbb{R}^C$ is a set of states. Each state $s_i$ is defined as a pair $(l_i, v_i)$ such that $l_i$ is a location and $v_i$ is a clock valuation for $C$ and $v_i$ satisfies $Inv(s_i)$, i.e., $S = \{(l, v) \in L \times \mathbb{R}_{\geq 0} | v \models \text{Inv}(l)\}$

$s_0 = (l_0, v_0)$ is an initial state, where $l_0$ is an initial location and $v_0 = 0$ for all clocks, and

$\rightarrow \subseteq S \times \{\mathbb{R}_{\geq 0} \cup E\} \times S$ is a transition relation that takes into account the invariants and that is defined as following:

- $(l, v) \xrightarrow{a} (l, v + d)$ if $\forall d' : 0 \leq d' \leq d \Rightarrow (u + d') \models \text{Inv}(l)$, where $d \in \mathbb{R}_{\geq 0}$.
- $(l, v) \xrightarrow{d} (l', v')$ if $\exists t = (l, e, \lambda, \varphi, l') \in T \Rightarrow v \models \varphi$ and $v' \models \text{Inv}(l')$, with $v' = [\lambda \mapsto 0]v$, where $[\lambda \mapsto 0]v$ denotes the clock valuation which maps each clock in $\lambda$ to 0.

B. Extended Time Unit System
The Time Unit System (TUS for short) [7] provides a simple way to specify time intervals and intervals based on a common way for expressing dates. TUS represents time intervals as sequences of non-negative whole numbers in the form [Year, Month, Day, Hour, Minute, Second,...]. For example, [2009, 6, 13, 22, 30, 0] denotes the first second of the minute 30 of the hour 22 on the 13th day of June 2009. Intervals can also be built using the operator convexity. For any two time intervals $t_1$ and $t_2$, $\text{convexity}(t_1, t_2)$ is the smallest convex interval that contains both $t_1$ and $t_2$. For example, convexity ([2000], [2009]) denotes the first ten years of the 21st century.

The Extended Time Unit System (XTUS for short) [5] extends TUS and improves its expressive power to define time intervals. In addition to the fixed time intervals presented as integers in TUS, XTUS proposes new elements. An interval element defines the time interval between two time instants, denoted by $[t_i, t_j]$ which is equivalent to $[t_i, t_i + 1, t_i + 2, ..., t_j]$. For example, $[5, 6, 7, 8, 9, 10, 11]$ is an anonymous element corresponds to any possible value of the time unit denoted by “.” (underscore – inspired from the anonymous variable in Prolog). For example, [2009, ...] denotes the first 10 days in any month of 2009. A combined element corresponds to the result of the combination of two different elements (i.e., integer or interval) in any level of hierarchy, denoted by {$.}$. For example, the combined element $\{[1-5], [8-12]\}$ denotes every day in every month except June and July (months 6 and 7) of any year.

XTUS supports also some algebraic operations for defining more complex temporal properties. Given $r_1$, $r_2$ two terms specifying intervals of XTUS, then $r_1 \cup r_2$, $r_1 \cap r_2$, $r_1 \setminus r_2$ denote respectively the union, the intersection, and the difference of these intervals.

C. AOP and AO4BPEL
Aspect-Oriented Programming (AOP) [2] is a programming paradigm, which supports the modularization of concerns that cut across the implementation of a software application, such as logging, persistence, and security.

Aspects are new units of modularity, which encapsulate crosscutting concerns in complex systems by using pointcuts, join points, and advice. Pointcuts are predicates over program execution actions – called join points. That is, a pointcut defines a set of join points related by some property; a pointcut is said to be triggered or to match at a join point, if the join point is in that set. An advice is a piece of code associated with a pointcut – it is executed whenever the pointcut is triggered, thus implementing crosscutting functionality. There are three types of advice, before, after, and around, relating the execution of advice to that of the action that triggered the respective pointcut.

AO4BPEL [3] is an aspect-oriented extension of the Web service composition language BPEL, which allows for more modular and dynamically adaptable processes. Like BPEL, AO4BPEL is XML-based. Aspects consist of one or several pointcut-and-advice The pointcut language of AO4BPEL is XPath, which is used to select activity joint points, i.e., points corresponding to the execution of activities). An advice is a BPEL activity that implements some crosscutting logic, e.g., security, monitoring, etc. Special constructs may be used inside the advice to access the input and/or output data of the join points as well as reflective information. Advice can be executed either before, after, or instead of (i.e., around) the intercepted BPEL activities. Further, an aspect may declare partners, variables, fault handlers, etc. An orchestration engine that supports the dynamic composition of AO4BPEL aspects is available.

D. Example: Travel Agency Scenario
We introduce an example Web service composition in the context of a travel agency scenario. In this scenario, we consider two business processes that contain activities representing either business tasks or interactions between Web services and customers: the travel package search process and the travel package booking process. Both processes compose services of airlines and hotels to find (resp. book) vacation packages whereby five partners are involved: the travel agency, customers, an airline, a hotel chain, and a payment service.

Once a customer sends a travel request to the travel agency, the travel package search process interacts with the information systems of the airline companies to find flights that match the customer needs. In addition, this process searches for available hotel rooms by interacting with the information systems of several hotels chains. After that, the travel agency sends the response as a set of offers combining the respective flight and hotel responses. Next, the customer could select one offer. This step launches the travel package...
booking process, which interacts with the airline and hotel services to verify availability. Upon availability, this process asks the customer to confirm the purchase of the travel offer and to provide payment details (e.g., using a credit card). In the last step, this process performs the booking of the different components of the travel package and sends a confirmation by e-mail to the customer, which includes all trip documents.

We present in the following some temporal constraints, which are important in the context of the travel agency example.

- **C1:** The customer should pay the selected offer by providing his credit card data within 30 minutes after the reservation step. Otherwise, the reserved offer will be canceled.
- **C2:** In a 5 minutes interval the customer can only do 3 failed payment trials.
- **C3:** The customer can cancel a travel reservation the latest 7 days before his travel.
- **C4:** The customer can change his travel reservation only 2 times. Changes are only allowed between 1 day and 5 days after the reservation date.
- **C5:** If the booking is done in a special period, a discount is given.

III. FORMAL SPECIFICATION

In this section, we first define the syntax and the semantics of our new formal language XTUS-Automata. Then, we introduce some specification patterns that enable expressing temporal constraints in Web service composition using XTUS-Automata.

A. The Formal Language: XTUS-Automata

To the best of our knowledge, there is no formal language that can be used to specify both relative time and absolute time properties for Web service composition. The combination of XTUS and timed Automata covers the limitation of time automata with respect to absolute time. In fact, XTUS allows to specify temporal properties that use absolute time, while timed automata allow to (1) specify the execution order of the different actions/activities in the process underlying the service composition, (2) specify the maximum and minimum occurred time between the execution of two different actions/activities, and (3) express that certain actions may be repeated $n$ times within a given time period.

In addition, temporal properties specified using XTUS-Automata can be easily translated to code. The translation of XTUS expressions is straightforward as time is specified using a set of integers and elements (i.e., interval, anonymous, and combined elements). Timed automata can be translated to code thanks to the specification patterns and the aspect templates, which will be presented later.

In addition to clock resets and to temporal constraints (see definition 2 in Sec. II-A), we propose to specify XTUS constraints at the automata transitions (see Sec. II-B). In this way, an action defined in the business process will be executed if and only if two types of constraints defined on the corresponding transition are satisfied: Timed Automata constraints and XTUS constraints. To allow expressing constraints that are relative to the current time, the designer can assume the availability of a special clock $ct$ that corresponds to the current time. That special clock cannot be reset.

In order to use XTUS for the formal specification of temporal constraints we have formally defined its syntax and semantics using the Z notation [8]. In the following, we present the formal definition of the different elements defined in XTUS (simple time or date value ($\mathbb{N}$), interval ($\text{Interval}([\mathbb{N} \times \mathbb{N}])$), anonymous ($\text{Anonymous}$), and combined elements ($\text{F UnitInterval}$), which were presented in Sect. II-B. The function $\text{Def}$ describes the semantics of each element: we associate for each type of element a set of integers that represents the possible values of this element. For example, the function $\text{Def}$ associates with the element $\text{Interval}(n_1,n_2)$ the set of all possible integers between $n_1$ and $n_2$.

$$\text{XTUS} = \text{seqElement}$$

$$\text{UnitInterval} ::= \mathbb{N} \mid \text{Interval}(\mathbb{N} \times \mathbb{N})$$

$$\text{Element} ::= \text{Anonymous} \mid \text{UnitInterval} \mid \text{F UnitInterval}$$


\[
\begin{align*}
\text{Def} : \text{Element} & \rightarrow \mathbb{N} \\
\text{Def}((\text{Anonymous})) & = \{n : \mathbb{N}\} \\
\forall n : \mathbb{N} \bullet \text{Def}(n) & = \{n\} \\
\forall n_1,n_2 : \mathbb{N} \bullet \text{Def}(\text{Interval}(n_1,n_2)) & = \\
& = \{n : \mathbb{N} | n \geq n_1 \land n \leq n_2\} \\
\forall u : \mathbb{N} \bullet \text{Def}(u) & = \{n : \mathbb{N} | n \in u\} \\
\cup \{\{n_1,n_2 : \mathbb{N} | \text{Interval}(n_1,n_2) \in u\} \\
& \land n \geq n_1 \land n \leq n_2 \bullet n\}
\end{align*}
\]

In our formal definition, we consider the time in the form [Year, Month, Day, Hour, Minute, Second] and a time instant is represented by a set of time values (natural numbers) corresponding to the different units. We specify also using appropriate \(\mathbb{Z}\) schemas the domains of the different time units (i.e., seconds, minutes, etc). The domains allow us to prohibit users from specifying constraints such as [2009, [1-4], [20-30]], as February does not have the days 29 and 30.

An XTUS constraint is satisfied only if the current time $ct$ is one of the possible time values corresponding to that constraint (after applying the function $\text{Def}$). For that, each time value in the current time instant $ct(i)$ (year, month,...) defined as natural number ($\mathbb{N}$) should satisfy its respective element.$X(i)$ ($\text{Element}$) in the XTUS constraints using the function $\text{Verif}$ defined below.

\[
\begin{align*}
\text{Def} : \text{Element} & \rightarrow \mathbb{N} \\
\text{Def}((\text{Anonymous})) & = \{n : \mathbb{N}\} \\
\forall n : \mathbb{N} \bullet \text{Def}(n) & = \{n\} \\
\forall n_1,n_2 : \mathbb{N} \bullet \text{Def}(\text{Interval}(n_1,n_2)) & = \\
& = \{n : \mathbb{N} | n \geq n_1 \land n \leq n_2\} \\
\forall u : \mathbb{N} \bullet \text{Def}(u) & = \{n : \mathbb{N} | n \in u\} \\
\cup \{\{n_1,n_2 : \mathbb{N} | \text{Interval}(n_1,n_2) \in u\} \\
& \land n \geq n_1 \land n \leq n_2 \bullet n\}
\end{align*}
\]

In our formal definition, we consider the time in the form [Year, Month, Day, Hour, Minute, Second] and a time instant is represented by a set of time values (natural numbers) corresponding to the different units. We specify also using appropriate \(\mathbb{Z}\) schemas the domains of the different time units (i.e., seconds, minutes, etc). The domains allow us to prohibit users from specifying constraints such as [2009, [1-4], [20-30]], as February does not have the days 29 and 30.

An XTUS constraint is satisfied only if the current time $ct$ is one of the possible time values corresponding to that constraint (after applying the function $\text{Def}$). For that, each time value in the current time instant $ct(i)$ (year, month,...) defined as natural number ($\mathbb{N}$) should satisfy its respective element.$X(i)$ ($\text{Element}$) in the XTUS constraints using the function $\text{Verif}$ defined below.
The satisfaction function $Sf$ allows to verify that the time values of all units (i.e., seconds, minutes, etc) in the current time instant are satisfied using the function $Verif$

$$
Verif_\text{-} : \mathbb{P}(\mathbb{N} \times \text{Element}) \\
\forall n : \mathbb{N}; e : \text{Element} \quad \bullet \quad Verif(n, e) \iff n \in \text{Def}(e)
$$

$$
Sf_\text{-} : \mathbb{P}(\text{seq} \mathbb{N} \times \text{seq Element}) \\
\forall \text{SeqNat} : \text{seq} \mathbb{N}; \text{SeqElem} : \text{seq Element} \\
\big| \#\text{SeqNat} = \#\text{SeqElem} = 0 \bullet \big| \\
Sf(\text{SeqNat}, \text{SeqElem}) \iff (\forall i : 1..\#\text{SeqNat} \bullet (Verif(\text{SeqNat}(i), \text{SeqElem}(i))))
$$

**B. Specification Patterns**

We propose specifying time related properties in Web service compositions using XTUS-Automata based on specification patterns, whereby a specification pattern is a generalized description of a commonly occurring requirement. The use of patterns facilitates the specification of complex temporal constraints and allows to structure the specification so that it can be translated automatically to runtime monitoring code.

In our specification model, the Web service composition is considered as a sequence of finite states, which are reached by transitions representing WS-BPEL activities such as the messaging activities invoke, receive, and reply used for invoking a partner Web service and/or interacting with a client. The message transmitted from the caller is denoted by the symbol $\text{msg}$ and the message received is denoted by the symbol $?\text{msg}$. For us only the time or date variables are considered as parameters in the transmitted messages $\text{msg}(t_1, t_2, \ldots)$.

The automata clock can be defined and initialized in three ways. First, the clock can be initialized to zero as in timed automata. Second, the clock can be an input parameter for a BPEL activity (i.e., defined as message transmitted between Web services). Third, the clock can be defined as an output parameter of some BPEL activity already executed.

In the following, we describe some XTUS-Automata patterns used to define some useful temporal properties that occur quite often in the context of Web service compositions.

**Patterns for duration properties**: These properties allow to specify a fixed duration between the execution of two different activities. An activity must eventually follow another activity within, after, or at the time $t$.

Figure 2 depicts three patterns of this type which specify that the activity $e_n$ should be executed within a fixed time after the execution of the activity $e_1$. In Fig. 2a, at the automata transition of the activity $e_1$, a clock $x$ is reset and a clock constraint is defined at the transition of the activity $e_n$. This activity $e_n$ can only be executed if the defined constraint is satisfied. In case the activity $e_1$ corresponds to a BPEL activity with a date or time as input parameter (see Fig. 2b), or if the BPEL activity returns a time or date as an output parameter (see Fig. 2c), these parameters are defined at the corresponding transition instead of the expression for resetting a clock. In these two cases, we use the current time $ct$ to define the clock constraints.

In Figure 3, we apply the pattern shown in Fig. 2a as duration pattern with clock reset for specifying the constraint C1 defined in Sect. II-D. C1 requires that the customer should pay the travel fees the latest 30 minutes after the reservation. We store the time of the reservation step by resetting the clock $x$ at the transition $\text{reservePack}$, and we define the clock constraint ($x < 30\text{min}$) at the transition $\text{bookTicket}$. In case the customer books the ticket of his travel before the end of the 30 minutes period, he will receive a confirmation of his booking; otherwise the reservation will be canceled automatically.

To specify the constraint C3, which states that the customer can cancel his reservation the latest 7 days before his travel, we apply the duration specification pattern with input parameter (Fig. 3). The transition $\text{reservePack}$ should be defined as a transition with input parameters (departure date $DD$ and arrival date $AD$). In the transition $\text{cancelReservation}$, the designer specifies the time constraint ($ct \leq DD - 7\text{days}$) to express that cancelling the reservation cannot be executed if it happens later than 7 days before the departure date.
Figure 3: Example of duration patterns

**Pattern for temporal properties over cardinalities:** These properties define that some activity $e$ can be executed successively $n$ times within a fixed time period.

Figure 4 shows the specification pattern for this type of temporal properties. The designer should specify the repetition of the activity $e$ using multiple transitions (and without using a loop). In the first transition, a reset clock has to be defined to store the time execution of this activity. The other ($n - 1$) activities should be defined as next transitions and can contain some clock constraints depending of the temporal property. In addition, between the transitions corresponding to the activity $e$, any other transition $a_i$ can be specified.

Figure 4: Temporal property over cardinalities

Figure 5 shows an instance of the pattern for temporal properties over cardinalities. This example pattern specifies the constraint C4 of Sect II-D, which states that a customer can change the date of his travel only two times and this must happen between 1 and 5 days after the first reservation. The designer should specify the resetting of the clock ($x = 0$) in the transition reserveP and define the clock constraint ($x > 1$ day) on the first changePack transition, and ($x < 5$ day) on the second changePack transition. Between these two transitions, the designer can specify other transitions that correspond to the booking of a new travel package and the reception of the confirmation email.

Figure 5: Second example of a temporal property over cardinalities

**Pattern for absolute time properties:** These properties allow to specify that an activity can be executed only if its current time satisfies some XTUS constraint, which is associated with a transition in the automaton. Note that this pattern can be easily combined with the two other patterns discussed above to specify more complex properties.

Figure 6: Absolute time properties

As example, we use this pattern for the specification of the constraint C5, which states that a discount is given if the booking is done in a special period (e.g. in the first week of March, April and May). The automaton that defines this constraint would look like the one shown in Fig. 7 where $e_i$ will be replaced by the transition bookTicket and the XTUS expression $\{[0], [3-5], [1-7]\}$ is specified on that transition.

Figure 7: Example of absolute time properties

IV. ASPECT-BASED RUNTIME MONITORING

After the specification phase, we automatically generate aspects from the formal specification to enforce the specified properties. For that purpose, an XSLT template-based generator engine was implemented as an eclipse plug-in. Our plug-in generates monitoring aspects in the AO4BPEL language [3] based on the defined templates and
the proposed patterns. Our approach is extensible as further patterns can be easily defined. The respective transformations to AO4BPEL aspects need to be defined using XSLT templates. These aspects will be deployed together with the processes on an aspect-enabled BPEL engine, which allows the dynamic composition of processes and aspects. For information on the performance of the AO4BPEL engine we refer the reader to [9], [3].

In the following, we first explain the aspect generation workflow. Then, we discuss the types of aspects generated for each specification pattern. Finally, we present in detail an aspect template that builds the basis for the aspect generation.

A. Aspect Generation Workflow

The designer uses our eclipse plug-in to formally specify his temporal properties using an XTUSA automaton based on the pre-defined patterns presented in Sect. III-B.

For a correct monitoring, certain constraints must be fulfilled by the automaton or the automata implementing the constraints. First, the automaton’s states and transitions must match the BPEL process implementing the composition. For example, the name of the transitions in the automata must match the names of activities in the BPEL process. Further, the name of the input and output parameters of the transitions must match those of the activities.

The designer can define either one XTUS automaton to specify the process and all temporal constraints or separate sub automata (i.e., one for each constraint). Once the XTUS-Automata are defined our eclipse plug-in exports them in XMI format and verifies their conformity with the BPEL process. If that step is successful we identify all instances of the specification patterns in the XTUS-automaton (or automata) describing the system and its temporal constraints. For each specification pattern, we defined incomplete and parameterized AO4BPEL aspects called aspect templates that enforce the property expressed by the corresponding pattern. An aspect template contains generic enforcement logic that is independent from a particular scenario.

After identifying all occurrences of the patterns, we generate automatically AO4BPEL aspects by filling the placeholders and parameters of the respective aspect templates with data from the concrete pattern instances. Still certain data and parameters need to be completed and this is done using the following two mappings:

- Each specified automaton transition is mapped to an XPath-expression [10] that matches WS-BPEL activities, i.e., in order to know when to reset or test a counter.
- Each temporal constraint from the automaton model is mapped to the expression language of WS-BPEL that is evaluated by a time helper service. The expression contains the concrete clock name and clock values.

Once the template is instantiated, the generated aspects can be deployed together with the process on the AO4BPEL engine. Thus, they ensure through runtime monitoring the enforcement of the respective constraint.

B. The Runtime Monitoring Aspects

We generate two types of aspects to enforce temporal properties specified as instances of the pre-defined specification patterns. The first type uses an after advice and allows to start a timer (timer aspect) or an activity counter (counter aspect) after the execution of a given activity. The second type uses an around advice (decision aspect) and allows or prohibits the execution of the intercepted activity according to the specified constraint and to the state of the timer and/or the counter defined in the first aspect.

Next, we explain in more detail which of these aspects are needed for each specification pattern, the mapping of constructs of the specification XTUS automaton to AO4BPEL pointcuts and advice, and the advice logic in each case.

Aspects for duration properties: The monitoring of duration properties requires a timer aspect and a decision aspect. The pointcut of the timer aspect intercepts the BPEL activity specified as a reset transition (i.e., the automaton transition labeled with a reset of the clock variable or the transition labeled with the definition of a clock as input or output parameters). The advice of this aspect invokes a helper time Web service to start a timer after the execution of the intercepted activity. For the decision aspect, the pointcut intercepts the execution of the activity specified as an automaton transition labeled with a temporal constraint. The advice of this aspect contains two parts. In the first part, a helper Web service is invoked to stop the timer before the execution of the intercepted activity. The identification of the timer in the two aspects is ensured by the name of the clock variable used in the two transitions specifying the intercepted activities. In the second part, the aspect checks the constraints defined in the automaton transition, which corresponds to the intercepted activity. If the constraint is satisfied, the activity will be executed. Otherwise, the aspect prohibits the activity and an exception will be raised.

Aspects for temporal properties over cardinalities: The monitoring of temporal properties over cardinalities can be seen as an extension of the monitoring of duration properties. In addition to the timer aspect, a counter aspect is generated, which allows to calculate the number of times an activity is executed. The pointcut of the counter aspect intercepts the BPEL activity specified by a transition that is repeated more than one time and defined between a reset transition and a transition that is labeled with a temporal constraint. The advice invokes a helper Web service to start and increment the counter. The decision aspect verifies also the number of times the activity is executed against the number of the transition labeled with activity name.
Aspects for absolute time properties: The monitoring of the absolute time pattern requires only a decision aspect. The generated pointcut intercepts the automaton transition labeled with an XTUS expression. The advice verifies if the current time and date satisfies the XTUS constraints as formally defined in Section III-A. Based on that the advice allows or prohibits the execution of the intercepted activity.

C. Aspect Templates

In the following, we present in detail an aspect template and illustrate the aspect generation based on the concrete examples shown in Sect. III-B.

Listing 1 shows the template of the timer aspects (lines 16–38) and the decision aspects (lines 44–73) defined in Sec. IV-A. As an example, we illustrate the use of this template to generate monitoring aspects for the duration patterns defined in Fig. 3. The template uses loop constructs to iterate other the set of transitions in the XTUS-Automaton specification (lines 14 and 40, 42 and 75) and placeholders (lines 18, 29, 46, 54, and 64).

For the timer aspect, the pointcut (lines 17–19) intercepts the corresponding BPEL activity specified by the reset transition. The Reset Transition placeholder (line 18) is replaced by a pointcut that matches the process activity that leads to the resetting of the corresponding counter. In an AO4BPEL aspect, the pointcut representing the interception of an activity is defined as an XPath-expression in the form of 
\[
\text{//@process//@reply[@operation="reservePack"]}
\]

For the example in Fig.3, the timer aspect intercepts the execution of the activity reservePack which resets the counter x. The Reset Transition placeholder is replaced by 
\[
\text{//@process//@reply[@operation="reservePack"]}
\]

After the execution of the intercepted activity, the advice (lines 21–37) invokes the operation startTimer (lines 4 and 34) on a helper time Web service to measure the execution time of the intercepted activity. This operation takes the name of the monitored activity and the clock variable as input parameters. The name of the activity is set by an assign activity (lines 23–32) to the name of the current join point activity, which is accessed by using the reflective AO4BPEL variable ThisJPActivity (line 25). The clock name placeholder (line 29) is extracted from the automaton transition corresponding to the intercepted activity.

For the monitoring of temporal properties over cardinalities (Fig. 4), an additional counter aspect should be generated with the same structure as the timer aspect. This aspect intercepts the corresponding activity and invokes another helper Web service to calculate the number of times the intercepted activity has been executed so far.

The decision aspect is (lines 44–73) defined with an around advice. The pointcut (lines 45–47) intercepts the activity modeled by the automaton transition labeled with the temporal constraint. The Enforce Transition placeholder (line 46) is replaced by a pointcut that matches the corresponding activity. For the example shown in Fig.3, the pointcut 
\[
\text{//@process//@reply[@operation="bookTicket"]}
\]

matches the activity representing the ticket booking, which should be performed in the first 30 minutes after reservation. In the advice (lines 49–72), a helper Web service is invoked to count the number of times the intercepted activity is repeated (line 59) and calculate the duration between this activity and the activity intercepted in the first aspect (line 61). At the end, the temporal constraint is verified using a switch condition (lines 63–70). The Constraint Expression placeholder (line 64) is replaced by the technical expression in the expression language of WS-BPEL. E.g., in Fig. 3 the temporal constraint \( x < 30 \) defined on the bookTicket transition is mapped into the expression “getVariableData(timeWsResponse,duration) > 30". If the constraint is satisfied, the corresponding activity will be executed using the keyword proceed (line 65), otherwise the activity will not be executed and an exception will be raised (line 67–69).

V. RELATED WORK

In this section, we present related works on the formal specification and the implementation of temporal properties in Web service compositions. Several formal methods were used to model Web services compositions such as Petri nets [11], finite automata [12], etc. The principal goal of these works is to verify the business process properties using model checking tools. They do not cover the specification and the implementation of non-functional properties such as temporal properties or quality of service.

Other works such as [13] proposed different formal languages and monitoring tools to specify and monitor service level agreements. Most of these works describe in a high-level specification the contract between the customer and the service provider in terms of quality of service properties such as performance and costs. These works do not address the temporal properties required in Web services composition.

Another group of related work concentrates on the specification of non-functional properties, and more specifically on temporal properties. In [14], the authors propose a new model, called web service timed state transition systems inspired from timed automata for specifying temporal behavior of web services compositions. The authors use also interval temporal logic to express complex temporal requirements. Based on this model, they propose a model checking approach for temporal properties on top of BPEL. The authors applied their approach to a complex e-government case study, but absolute time properties cannot be specified. They also do not propose any enforcement mechanism.

Baresi et al. [15] propose a temporal extension of an XML-based assertion language called Timed WSCoL, which allows expressing temporal properties over the actions that
Listing 1: AO4BPEL Aspect Template
occur during the process execution. The authors propose powerful operators (such as between and within) in addition to the linear temporal operators (such as always and until). For the monitoring phase, they propose to use an extension of an aspect-oriented language called Dynamo and an external analyzer to ensure asynchronous monitoring. This approach does not cover all temporal properties addressed in this paper namely the properties defined by XTUS. Also this work does not provide any mechanism to easily specify complex time-related properties.

The approach proposed by Barbon et al in [16] is quite similar to ours. The authors propose the specification of boolean temporal properties and statistic properties in BPEL processes using the RTML language (Runtime Monitoring Specification Language). They also generate Java code for monitoring the specified properties. However, compared to our approach, the work presented in [16] is less expressive than ours as absolute time properties for example are not supported. Further, the monitoring code generated in [16] is not modular as it is mixed with the application code. In addition, that approach is static i.e., it does not allow dynamic changes of temporal properties at runtime, which is possible in our approach thanks to dynamic weaving in the AO4BPEL engine.

In [17], the authors specify temporal properties using an extension of the WSTTS language. They use temporal properties in order to analyse the compatibility in Web service composition. Compared to that proposed approach, we use a similar model to specify temporal properties in Web service composition because we used an extension of timed automata. However, our approach supports the relative and absolute time in the formal specification. In addition, in [17], no monitoring mechanism is presented.

VI. Conclusion

In this paper, we presented a holistic approach to temporal constraints in Web service compositions, which covers specification and monitoring. For the specification, we introduced a novel formal language, which combines Timed Automata and XTUS. We also presented several specification patterns which should be followed to specify various types of temporal properties. Our approach allows a separate specification of the temporal constraints, which are no longer intertwined with the BPEL process that defines the composition. Further, our approach enables the formal verification of temporal constraints and the detection of possible inconsistencies. This is not possible when temporal constraints are implemented with BPEL activities. In addition, our approach provides mechanisms for the monitoring of the temporal properties through the automatic generation of aspects.

As future work, we will extend our approach by other specification patterns for more complex temporal properties and will implement the necessary templates for generating the monitoring aspects.

REFERENCES