HOW TO TRACK COMPOSITE WEB SERVICES? A SOLUTION BASED ON THE CONCEPT OF VIEW

Djamal Benslimane
Claude Bernard Lyon 1 University
Lyon, France
djamal.benslimane@liris.cnrs.fr

Zakaria Maamar
Zayed University
Dubai, United Arab Emirates
zakaria.maamar@zu.ac.ae

Chirine Ghedira
Claude Bernard Lyon 1 University
Lyon, France
chirine.ghedira@liris.cnrs.fr

ABSTRACT

This paper presents an approach for tracking composite Web services based on the concept of view. Web services are subject to personalization when there is a need to accommodate user preferences. Preferences of users are of various types like when the execution of a Web service should be initiated and where the outcome of this execution should be delivered, too. To guarantee that user preferences are handled during the execution of Web services, a view offers the opportunity of zooming into the specification that composes these Web services. For the specification requirements, state chart and service chart diagrams are adopted. A deployed view over a progressing specification features the dynamic nature of tracking composite Web services.

Keywords: Composition, Context, View, Web Service.

1 Introduction

1.1 Motivation

Web services technology offers, nowadays, a major support for a new generation of information systems featured by different components that collaborate despite distribution and heterogeneity obstacles (Schahram and Schreiner, 2005; Chen and Meixell, 2003; Chen et al., 2003; Kwok and Chi, 2006). A Web service is a software component that other software components and humans can discover and trigger to satisfy different needs (e.g., weather forecasts). Several standards back the development of Web services for their discovery, description, and binding, respectively (Milanovic and Malek, 2004). Taking advantage of the extensible Markup Language (XML) in order to achieve their platform-independence feature, Web services also have the capacity to be composed into high level business-processes usually referred to as composite services. Composition primarily addresses the situation of a user request that cannot be satisfied by any available Web service, whereas a composite service obtained by integrating available Web services might be used (Berardi et al., 2003). Composing services rather than accessing a single service is essential and offers better benefits to users. For composition requirements, a composite service is associated with a specification that describes among others the list of component Web services that participate in the composite service, the execution order of the component Web services with respect to data and temporal dependencies, and the corrective strategies in case of exceptions. There exist multiple languages for the specification of Web services compositions including the Web Services Flow Language (WSFL) (Leymann, 2001) and the Business Process Execution Language (BPEL) (Curbera et al., 2003). The primary aim of these specification languages is to provide a high-level description of the composition process far away from any implementation concerns. The specification of composite services is also concerned with the semantics of information that the component Web services exchange (Sabou et al., 2003), but this is outside this paper's scope.

While much of the R&D on Web services have focused on low-level standards for publishing, discovering, and invoking Web services (Ma, 2005), it is deemed appropriate tracking the execution of Web services. However, *very*

little has been achieved to date regarding this tracking due to several obstacles including: current Web services are not active components that can be embedded with context-awareness mechanisms, existing specification approaches for Web services composition typically facilitate choreography only while neglecting context and its impact on this choreography, and lack of guidelines supporting the operations of Web services tracking. One of the advantages of this tracking is the possibility of adjusting Web services according to user preferences and constraints over the environment (Maamar et al., 2005). Indeed, some users would like receiving answers to their personal requests directly submitted to their personal email instead of office email. Personalization is tightly related to the features of the environment in which the Web services are to be executed once they are triggered. These features can be related to users (e.g., stationary, mobile), user level (e.g., expert, novice), computing resources (e.g., fixed device, mobile device), time periods (e.g., in the afternoon, in the morning), physical locations (e.g., mall, cafeteria), etc. Sensing, gathering, and refining the features and changes in an environment allow the definition of what is known as *context*. Context is the information that characterizes the interactions between humans, applications, and the surrounding environment (Brézillon, 2003). Embedding services whether Web or composite with context-awareness mechanisms has several advantages as Maamar et al. report in (Maamar et al., 2005: Maamar et al., 2006).

1.2 Contributions

Context-aware computing refers to the ability of a software application to detect and respond to changes in its environment (Roman, 2002). In this paper, the main use of context is to identify and adjust the specification of a composite service according to the current features of the environment. To identify which part of the specification of the composite service has to be *adjusted* because of the changes in the environment surrounding user, an assessment of what-was-previously-expected *vs*. what-is-effectively-happening is required. We refer to this part of the specification of the composite service as *view*. The assessment of "expected" *vs.* "happening" enables detecting any discrepancy so corrective measures are promptly taken. We define *a view as a dynamic snapshot over the specification of a composite service according to a certain context.*

Figure 1 illustrates a view that associates the context of user with a composite service specification. In this figure, the view comprises WS_2 and WS_3 as component Web services. It should be noted that what-was-previously-expected reflects the preferences of users towards the deployment of some Web services in terms of execution time, execution location, etc. And what-is-effectively-happening reflects the capacity of satisfying the preferences of users according to what currently exists in terms of computing resources, communication networks, etc. In this paper we discuss the way we use views as a means for first, tracking the execution progress of Web services and second, deploying corrective measures in case of non-compliance with the requirements of users. Our contributions are multiple: a clear definition of what a view means in the context of Web services composition, mechanisms for assessing the discrepancies between what-was-expected and what-is-happening, an approach to specify user context and its respective view, and mechanisms for extracting and visualizing views over specifications of composite services.

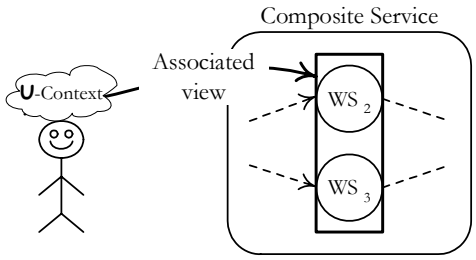

Figure 1: Representation of a service chart diagram

1.3 Paper organization

The rest of this paper is organized as follows. Section 2 overviews the Web service model, motivates the adoption of views with a running scenario, discusses the rationale of context and views, and finally presents some related works on views. Section 3 is about the foundations and principles of a view. Section 4 presents the formal model of a view. Section 5 discusses how the outcome of running a view over a composite service specification is subject to another view. The specification of user context and description of the proof-of-concept prototype is reported in Section 6 and Section 7, respectively. We draw our conclusions in Section 8.

2 Background

2.1 The Web service model

The Web service model comprises three entities: provider, registry, and consumer. The provider develops Web services and describes them in a standard way using WSDL (Web Services Definition Language). Afterwards it announces the Web services in a central registry, usually known as UDDI (Universal Description, Discovery, and Integration) so potential consumers can locate them according to their needs. When a consumer retrieves a specific Web service, it binds to it using an invocation protocol like SOAP (Simple Object Access Protocol). In (Dustdar and Schreiner, 2005), the six different issues that impact Web services composition were reported: coordination, transaction, context, conversation modeling, execution monitoring, and infrastructure. WS-CAF standing for Web Services Composite Application Framework is a Sun Microsystems initiative for dealing with coordination, transaction, and context issues (Bunting et al., 2003).

2.2 Running scenario

Our scenario concerns Melissa, a tourist who just landed in Dubai. Melissa has a PDA and plans to use it as a tourist *mobilebook* instead of carrying brochures. Upon arrival to the airport she decides on downloading some applications into her PDA as the tourism authorities offer those free-of-charges. These applications constitute the front-ends of Web services that concern tourism in Dubai.

Melissa picks *sightseeing* and *shopping* applications. Melissa is notified that these applications can be combined according to a certain composition pattern, and she decides on doing so. Melissa's plans are to visit outdoor places in the morning and go shopping in the afternoon. The first part of the plan is subject to weather forecasts as outdoor activities are cancelled in case of hot weather. Initially Melissa is prompted among other things to select some outdoor places, indicate the pickup/drop-off times for sightseeing and shopping, and express her need for a guide during the visits.

Once Melissa's preferences are set, they are submitted to appropriate Web services for processing. With regard to the first activity, *sightseeing* Web service checks with *weather* Web service the forecasts for the five coming days[1]. If there is no hot-weather warning, the scheduling of the places to visit starts by ensuring that these places are open for public on these days, and transportation and guide are arranged. The logistics of Melissa's multiple rides is affected to *transportation* Web service, which identifies for instance the type of vehicle, and the possibility that Melissa shares rides with other tourists heading towards the same places. In case of hot-weather warning, *sightseeing* Web service might suggest different places (e.g., museums) where indoor activities can take place. Same considerations apply to the shopping activity, which consist of checking out the running promotions in the malls that Melissa has selected. If Melissa has included several malls in her shopping plan, the distance between them is considered so appropriate transportation is scheduled, too. *Transportation* Web service is responsible for coordinating the timings among all the activities. Once the specification of Melissa's plan is finalized, a light version of this specification is submitted back to her PDA for records. By light version we mean the following details: list of participating Web services, their execution chronology, and approximate duration of the planned activities. Extra details can be added to the light version of a specification if needed.

The day after her arrival, Melissa is given a ride to an historical site. Due to an unexpected traffic jam, her PDA (in fact a software component residing in the PDA and acting on Melissa's behalf (Boudriga and Obaidat, 2004)) compares her current location to the location in which she is supposed to be (i.e., historical site) at that time. Noticing that Melissa is not in the agreed location, her PDA takes actions by sending a note to *sightseeing* and *transportation* Web services so corrective measures are performed like informing the guide about the delay.

The aforementioned scenario yields insight into the multiple challenges that tracking Web services faces. Some of these challenges include: what are the various reasons that initiate contacting Web services, is the location of a person sufficient for tracking the execution of the specification of her plan, what are the arguments (e.g., location and expertise) that could populate user context, how much does context of Web services contribute to this tracking, how to identify the component Web services that constitute a view executed over a specification, what are the relevant mechanisms that enable extracting and visualizing a view, and what are the features that a specification language for Web services composition (e.g., BPEL) has to have so a view happens with less interruptions to the execution of a specification?

2.3 Rationale of context

Despite the wide embracement of Web services, they still lack the capability that could propel them to the acceptance level that features traditional integration middleware such as CORBA and DCOM. This lack of capability is primarily due to the trigger-response exchange pattern that is imposed on Web services and their interaction models with the external environment whether peers, users, information sources, etc. The compliance

[1] In a previous research, Web services interactions were dealt with as conversations (Maamar et al., 2005).

with this pattern means that a Web service needs only to process the requests it receives, without for example considering its execution status, or even questioning about the validity of these requests. However, there exist several situations that call for Web services self-management so that flexibility, autonomy, and stability requirements are met. By flexibility, we mean the capacity of a Web service to adapt its behavior by selecting the appropriate operations that accommodate the ongoing situation (e.g., negotiation, monitoring) in which it operates. By autonomy, we mean the capacity of a Web service to accept demands of participation in composite services, rejects such demands in case of unappealing rewards, or eventually propose other alternatives for its participation by recommending some peers. And, by stability, we mean the capacity of a Web service to resist changes while maintaining function and recover to normal levels of function after a disturbance. To meet these requirements, Web services have to assess their environment prior they engage in any composition. In fact, Web services need to be context-aware.

2.4 Rationale of views

In the three-level ANSI-SPARC architecture, a view corresponds to the structure of the database as it appears to a particular user. In the relational model, a view is defined as a virtual relation that is dynamically derived from one or more other base relations. The concept of view is also used in the workflow community. Workflow view was suggested as a support mechanism for the interoperability of workflows across multiple businesses (Chiu et al., 2004).

Various reasons motivate our adoption of views for tracking Web services. First, the view mechanism provides a powerful and flexible security approach by hiding the complete specification of a composite service from users and the process responsible for adjusting this specification. Only the relevant parts of a specification, which are subject to adjustment, are presented. Second, because the same specification of a composite service is offered to several users for triggering, multiple views over this specification can be obtained at different levels of granularity ranging from the dependency between Web services and the execution preferences associated with Web services to the corrective measures that Web services use. Finally, a view helps propose criteria like security and visualization that specification languages for Web services composition have to include.

2.5 Some related works

There are several research initiatives in the field of Web services (Schahram and Schreiner, 2005). However, to our knowledge, none of these initiatives have attempted using views for tracking the deployment of personalized Web services. In the rest of this section, we highlight some of the works that have supported our thoughts and inspired shape our view-based tracking of personalized Web services.

An experience reporting the use of views in Web services is presented in (Fuchs, 2004). Fuchs adopted views to support the adaptability of Web services in a heterogeneous environment. Views enable service providers to segment their offerings in a logical fashion and provide specialized WSDL definitions for different potential groups of users. To this end, a service view describes the allowed operations and where to communicate with these operations. According to Fuchs, the service view fills in an important gap in Web services development by supporting the variety of interaction patterns across all the set of partners who may use specific aspects of the operations involved in the service.

In (Liu and Shen, 2003), Liu and Shen claimed that given the diverse requirements of the participants involved in a business process, it is critical to provide these participants with adequate process information. Their solution uses a process view, which is an abstracted process that is derived from a base process for modeling virtual workflow process. As a result, participants are provided with various views of the same process, thanks to an automatic generating tool.

In (Pistore et al., 2004), the authors address the issue of Web services composition planning and monitoring. Pistore et al. propose an approach based on the "planning as model checking" techniques in order to overcome difficulties related to planning under uncertainty. These difficulties are non-determinism, partial observability, and extended goals.

The notion of view is adopted by Roman et al. (Roman et al., 2002), who suggested a declarative approach to agent-centered context-aware computing in ad-hoc wireless environments. Because context, in (Roman et al., 2002), is seen as fine-grained units referred to as views, a view constitutes a projection of the maximal context together with an interpretation that defines the rules of engagements of a software agent in a particular view. The maximal context includes information that all hosts in the network store and the context information (e.g., location) that the agents sense on these hosts.

Nassar et al. proposed VUML, standing for View-based Unified Modeling Language, as a means for supporting the concept of multiview class (Nassar et al., 2003). VUML's objective is to store and deliver information according to users' viewpoints. Nassar et al.'s proposal consists of a base class (i.e., default view) and a set of views that correspond to different viewpoints obtained over the base class. Based on user's profile, a view-extension

relationship is set between a view and an extension of the default view.

3 Foundations of views

Before we discuss the foundations of view, we overview first state/service chart diagrams as a means for specifying compositions. A specification of Melissa scenario is also given in this section.

3.1 State/Service chart diagram

A state chart diagram is a graphical representation of a state machine that visualizes how and under what circumstances a modeled element changes its states (Harel and Naamad, 1996). Furthermore, a state chart diagram shows which activities are executed as a result of the occurrence of events.

A service chart diagram is a means for modeling and specifying the component Web services of a composite service (Maamar et al., 2003). It enhances a state chart diagram, putting now emphasis on the context surrounding the execution of a Web service rather than only the states that a Web service takes (Figure 2). To this purpose, the states of a service are wrapped into five perspectives. The *state* perspective corresponds to the state chart diagram of the service. The *flow* perspective corresponds to the execution chronology of the composite service in which the service participates. The *business* perspective identifies the organizations that offer the service and post it on an UDDI registry. The *information* perspective identifies the data that are exchanged between the service and the rest of the services of the composite service. Finally, the *performance* perspective illustrates the ways of invoking the service. More details on service chart diagrams are given in (Maamar et al., 2003).

When it comes to Web services personalization, users indicate when and where they would like to have these Web services performed according to particular periods of time and particular locations (additional types of preferences could be added). To illustrate user preferences, ***location*** and ***time*** perspectives are anchored to a service chart diagram (Figure 2). Location/Time identifies particular places/time periods (e.g., classroom/at 8am). In this paper, the semantics of time and location perspective is based on the work of (Allen, 1983) and (Papadias and Sellis, 1994), respectively.

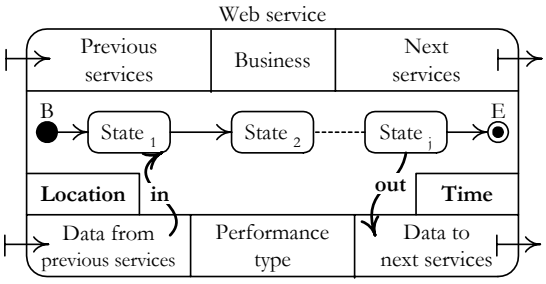

Figure 2: Combination of context and view

Because a composite service is made up of several component services, the process model that underlies this service is specified as a state chart diagram (the value-added of state charts to Web services composition is discussed in (Benatallah, 2002)). In this state chart diagram, states are associated with the service chart diagrams of the component services (Figure 2), and transitions are labeled with events, conditions, and variable assignment operations. Figure 3 is the specification of the composite service of Melissa scenario. Each component service is associated with a service chart diagram: sightseeing (**SI**), weather (**WE**), shopping (**SH**), and transportation (**TR**). These service chart diagrams are connected through transitions; some of these transitions have constraints to satisfy (e.g., [confirmed (hot weather)]).

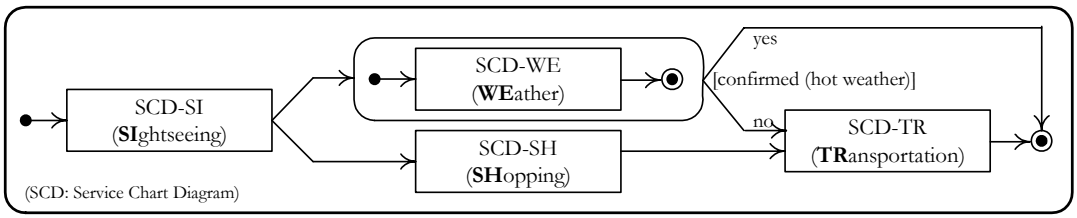

Figure 3: Specification of the composite service of Melissa scenario

3.2 The view meta-model

Figure 4 illustrates the meta-model of view for tracking Web services. This meta-model revolves around two types of concept. The first type of concept identifies the building blocks of any context-aware and Web services-based system namely: context, Web service, and composite service. The second type of concept identifies the mechanism of running context-based requests over specifications of composite services. The view implements this mechanism. A view is represented with a rounded-corner rectangle in Figure 4, so it is differentiated from Web service and composite service concepts that are represented with regular rectangles.

Context is of three types: user (U-context), Web service (WS-context), and composite service (CS-context). At this stage of our research, we only focus on views associated with U-contexts of users. The connection (context, user/Web service/composite service) is highlighted in Figure 4 with *related to* edges. U-context enables tracking user in terms of current location, current activities, etc. WS-context refers to the current capabilities and ongoing participations of a Web service in concurrent compositions. Finally, CS-context informs about the execution status of the component Web services of a composite service.

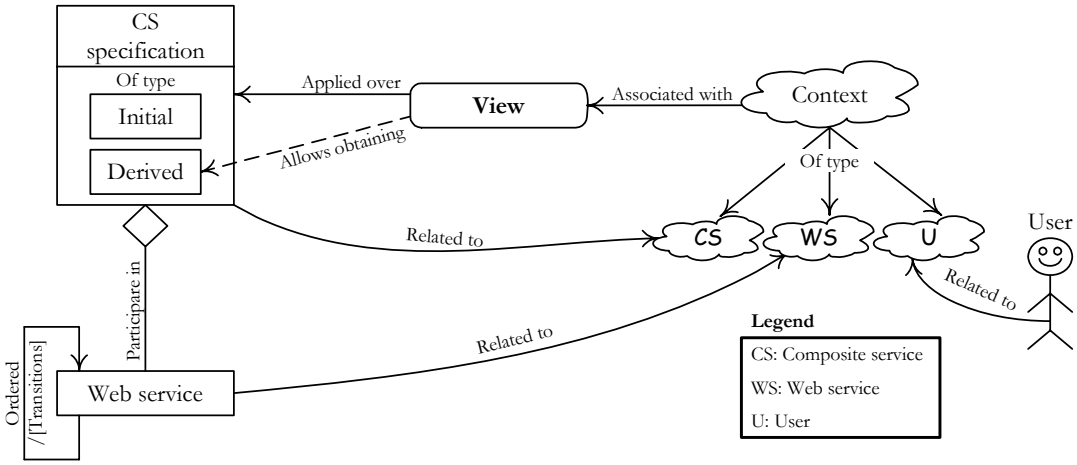

Figure 4: Representation of the view meta-model

Similar to context, a composite service specification is decomposed into two types: *initial* and *derived*. An initial specification is the specification that a designer devises for the first time, and in which he describes for instance the chronology of the component Web services of a composite service and the types of dependencies between them. Figure 3 is a composite service specification of type initial. A derived specification is obtained after running a view over a specification. This specification might be either of type initial or derived. To illustrate these two types of specification, the connection (view, composite service specification) is highlighted in Figure 4 with two different edges: *applied over* as solid edge and *allows obtaining* as dashed edge. Therefore, a derived specification can also be subject to another view operation. The capability of reusing the derived specifications of composite services is another argument in favor of adopting views for Web services tracking as motivated in Section 2.3. Finally, the aggregation of Web services into composite services is highlighted in Figure 4 with *participate in* edge. This participation complies with a specific execution chronology that corresponds to *ordered/[transitions]* line. [Transitions] represents the conditions of connecting Web services together.

The dynamic aspect of the view meta-model consists of two steps. The first step checks the transitions that have constraints (e.g., [checked(walking distance)], Figure 3) before any view is applied over a specification. Checking transitions limits the number of Web services of a composition specification to be involved in a view. Once these constraints are satisfied because of the features of the current context, the second step identifies the parameters that will be included in the extraction of a view from a composite service specification. Two parameters are identified: execution time and execution location.

4 Formal model of a view

In the previous sections, we illustrated how a view is dependent on the preferences of users and constraints that frame the transitions between Web services. Execution time and execution location are examples of preference, and weather forecasts and walking distance are examples of constraint. This section presents a formal specification of the view concept and proceeds next with an application of this formal specification to Melissa scenario.

4.1 Definitions

Definition 1 *- State chart diagram*

Let ISCD and DSCD be the set of Initial and Derived State Chart Diagrams, respectively. And, let SCD be the set of all State Chart Diagrams that specify composite services. SCD= SCD $\cup$ DSCD. A state chart diagram scd $\in$ SCD is a triple scd=<S,T,Tc> where:

- S={s_1, s_2,…} is the set of temporal and localized service chart diagrams. $\forall$ s $\in$ S, s=f_s(t,l) where t and l represent time and location parameters, respectively.
- T={t_1, t_2,…} is the set of unconstrained transitions.
- Tc={tc_1, tc_2,…} is the set of constrained transitions (e.g., is the weather hot?).

Definition 2 *- Context*

Let CONT be the set of all potential contexts that can be generated after data collection (e.g., from sensors) and refinement. A context cont $\in$ CONT is associated with the function F(U, W, C) where U, W, and C represent User, Web service, and Composite service contexts. As mentioned earlier, only U-context is used for extracting derived state chart diagrams. U-context is defined by the function $f_U(arg_1, arg_2,\ldots)$, where arg_i is a user parameter such as status and location.

In the following we formally describe through the concept of derived state chart diagram, how a view is extracted from a given scd according to a given context cont.

Definition 3 *– Derived state chart diagram*

Let scd and cont be a state chart diagram (initial or derived) and a context. A derived state chart diagram dscd $\in$ DSCD is a tuple dscd: View(scd, cont)=<S',T',Tc'> where:

- S' $\subseteq$ S.

  This means that a derived specification does not accept any additional services through their respective service chart diagrams. However, some existing elements like states and transitions of S could be excluded from the derived state chart diagram. For instance if the constraints on an incoming transition of a service chart diagram are not satisfied in a certain context, then this service chart diagram will be excluded from the derived state chart diagram.

- T'={t' | either t' $\in$ T $\wedge$ InitialState(t') $\in$ S' or $\exists$ tc $\in$ Tc | t'=FullInstanciation(tc,cont) $\wedge$ InitialState(t') $\in$ S'}.

  InitialState function determines the initial state of a transition. FullInstanciation function returns an unconstrained transition when the constraint on this transition is satisfied in the current context cont. Therefore, the unconstrained transitions in T' of a derived state chart diagram dscd are obtained either from (1) the unconstrained transitions of scd for which the initial state chart diagram belongs to S', or (2) the constrained transitions of scd that are satisfied in the current context cont.

- Tc'={tc' | tc' $\in$ Tc $\wedge$ UnSatisfied(tc', cont) = false $\wedge$ InitialState(tc') $\in$ S'}.

  Tc' is the set of constrained transitions tc' that are unknown (i.e., neither satisfied nor unsatisfied) in the current context cont. Plus, the initial state of these constrained transitions is an element of S'. UnSatisfied function checks whether a transition is satisfied in the current context.

4.2 Application to Melissa scenario

Figure 3 represents $scd_{Melissa}$, the state chart diagram of Melissa scenario. The diagram is a triple <S,T,Tc> where S={scd-si, scd-we, scd-sh, scd-tr}, T={t_0, t_1, t_2, t_{11}}, and Tc={t_7, t_8, t_9, t_{10}}. Let us assume that Melissa's context returns details on weather conditions and walking distance between malls: cont = ((confirmed(hot weather)=yes $\wedge$ checked(walking distance)=yes). Figure 5 is the derived state chart diagram dscd that is extracted from $scd_{Melissa}$ for this given context cont. dscd is defined by a triple <S',T',Tc'> where S'={scd-si, scd-we, scd-sh}, T'={t_0, t_1, t_2, t_7, t_{10}} and Tc'=Ø.

The constrained transitions t_7 and t_{10} in scd turn out unconstrained in dscd. Their respective constraints are satisfied in the current context (cont =(confirmed(hot weather) = yes $\wedge$ checked(walking distance) = yes)). Not satisfied in the current context cont are the unconstrained transitions t_8 and t_9, which are excluded from dscd. In general a derived state chart diagram is in a constant evolution along with the dynamic nature featuring the user context (Figure 6). For instance, constrained transitions become unconstrained when more information about the user's context are made available. In the case of Melissa, the obtained derived-state chart diagram is qualified as final; all its transitions are unconstrained. The current time and location values will permit the detection of the Web services that are under execution. This is completed using time and location perspectives of the service chart diagram.

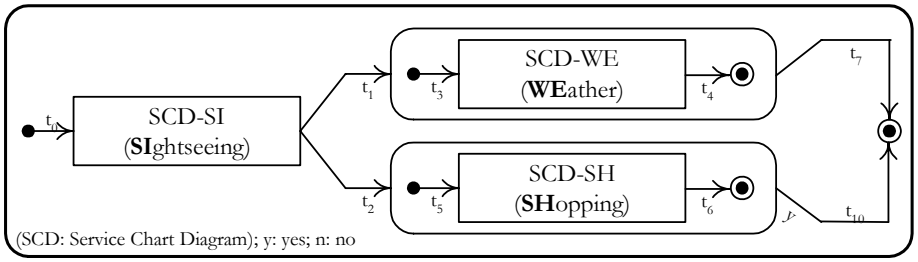

Figure 5: Sample of a derived state chart diagram

Figure 6 illustrates the progress of a view as time advances, location changes, or constraints on the environment are satisfied. U-context$_1$ and U-context$_i$ represent past and current context of user, respectively. Each context is associated with a view. "Applied over" layer in Figure 6 corresponds to the mechanism that supports running a view over a composite service specification. Since a view is context-dependent, a view is tightened to the constant evolution of context. This evolution is backed by Lonsdale and Beale (Lonsdale and Beale, 2004). They consider context not as a static phenomenon but as a dynamic process, where context is constructed through the learner's interactions with the learning materials and the surrounding world over time. Mobile learning is the application domain of Lonsdale and Beale's project.

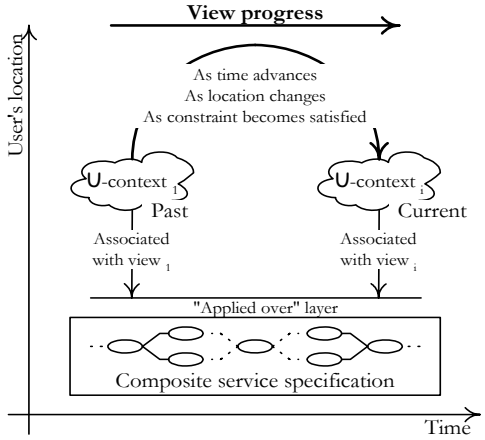

Figure 6: Representation of a view progress

5 Management of derived composite service specifications

In Figure 4, the specification of a composite service was specialized into two types: *initial* and *derived*. We positively argued that a view could be applied over a derived specification, which could have been attained either from an initial or another derived specification. The capacity of reusing derived specifications was another argument in favor of adopting views as a mechanism for tracking Web services.

It is worth noting that since the last time a derived state chart diagram was generated (e.g., Figure 5), the environment has definitely changed like new content for context, new time period, new location, etc. This means that the derived state chart diagram has to be reviewed before it can be used in another view operation. This probably means adding/discarding transitions and states to/from this derived state chart diagram, which results in a new derived state chart diagram.

In order to maintain a consistent derived state chart diagram when the context cont is updated, we have developed a set of procedures that permit *reflecting* the instantaneous changes on a derived state chart diagram. An example of change could be a transition that is now valid in the current context, but was invalid in the previous context. These procedures are based first, on associating four *colors* (red, blue, green, and black) with transitions of a state chart diagram and second, on mapping this diagram onto a graph where states and transitions correspond to nodes and edges, respectively.

- Let AN be a set of accessible nodes. An accessible node is a state of the derived state chart diagram that is reachable using the transitions which are not declared as unsatisfied (either satisfied or unknown) in the current context cont.

- Different colors are associated with a transition t:
  - t is blue *if and only if* t is satisfied and InitialState(t) $\notin$ AN; i.e., t is satisfied but its input state is not yet reachable.
  - t is green *if and only if* t is satisfied and InitialState(t) $\in$ AN; i.e., t is satisfied and its input state is reachable.
  - t is red *if and only if* t is unsatisfied.
  - t is black *if and only if* t is neither satisfied nor unsatisfied. This happens when enough details on t are missing from the context.
- When the context cont changes of content, the state(color) of some transitions is affected (i.e., a transition which was satisfied becomes unsatisfied, etc.).
- After updating the color of a transition, other transitions can also be affected because of this update (i.e., a *green* transition can change color to *blue* if its previous transition becomes *red*, etc.).

To manage the change in the color of transitions and the reflection of this change on the whole derived state chart diagram, we defined a set of procedures (Appendix 1). These procedures are responsible for adding/discarding states and transitions to/from a state chart diagram according to the information that the current content returns. For illustration purposes Figure 7 presents *Red_To_Blue* procedure. It is executed when a transition t becomes satisfied after adding new elements to the context cont.

Once the procedures of Appendix 1 are executed according to the content of a given context, the new derived state chart diagram on which a view will be performed, is represented in the graph by the path connecting states using only transitions of green and black colors. The rest of the graph corresponds to the specification of the composite Web service, which is irrelevant to the content of the current context.

```
Proc Red_To_Blue(t)
Input: t Red Transition
Begin
// Executed when the unsatisfied transition t becomes satisfied
t.Color ← Blue // T moves to Bleu
in ← InitialState(t)
on ← Out_State(t)
// Check whether the new satisfied transition t can be added to the dscd diagram and then
propagate this change
if in ∈ {AN} and on ∉ {AN} then
        AN ← AN ∪ {on }
        t.Color ← Green
        for t' ∈ Ongoing_Transitions (on) do
                If Color (t') == Blue then Blue_To_Green(t')
                end if
    end for
end if
End
```

Figure 7: Red_To_Blue procedure

6 Definition of user context

We have shown that applying a view over a composite service specification depends on the relevance and quality of the information that U-context caters for this view process (Figure 1). In what follows we detail the internal structure of U-context. We rely, to a certain extent, on our previous work (Ghedira et al., 2002). Prior to that we list the objectives that are to be reached by using U-context: provide a traceability means of the actions that a user has executed in the past, is currently executing, and might be executing in the future; consider the changes in a user's needs; and classify and present information on user long-term properties including her role in the society (e.g., travel agent), her status abroad (e.g., tourist), etc.

Based on the aforementioned objectives, we organize U-context along a set of attributes where each attribute is linked to a table-based data structure. These attributes are of different types and have different potential applications. Therefore, we classify the attributes according to the dynamic aspect of the information that features a user. For instance user location always changes while user gender is to a certain extent static. In addition the interests of a user are also added to his respective context so that appropriate proposals are made. If Melissa is on sports avenue

($user_{location}$ ← GET(U-context(Melissa).*location*)) and the top sightseeing place on this street is Dubai's soccer stadium, the gender of Melissa ($user_{gender}$ ← GET(U-context(Melissa).*gender*) could help propose Dubai's national museum situated two streets down Melissa's current location instead of the soccer stadium. This constitutes an attractive proposal for Melissa if she sets the attribute of her interest in arts to very high.

The classification of U-context's attributes has two parts. The first part concerns the long-term attributes[2] and the second part concerns the attributes of the environment that surrounds the user. The *Long-Term Attributes* (LTA) of U-context are as follows (U-$context_{LTA}$ = <U_1,U_2,U_3>).

- U_1 is about the user's personal characteristics in terms of age, social status, etc.
- U_2 is about the user's interests in certain topics including name, level of interest, etc.
- U_3 is about the user's perceptual and motor skills and limitations as well. The capabilities of his devices[3] (e.g., Melissa's PDA) are also included in U_3.

With regard to the *Attributes of the Environment* (AE) of U-context they are as follows (U-$context_{AE}$ =<U_4,U_5>).

- U_4 is about the user's situation in terms of current location and ongoing activities. A location may be obtained from sensors or advanced positioning techniques.
- U_5 is about the user's current state. This information is more complex to conceptualize as it refers to cognitive and psychological aspects.

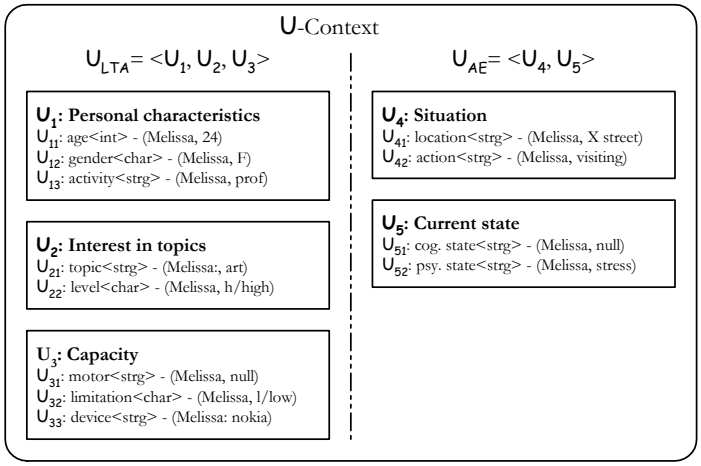

Figure 8: Components of U-context

As Figure 8 illustrates, The integration of U-$context_{LTA}$ and U-$context_{AE}$ into U-context is at this stage of our work sufficient to support the generation of views over specifications of composite services. One of the benefits of associating users with U-contexts is the possibility of generating *trails*, which capture for each user his daily activities. Clarke and Driver define a trail as a collection of locations together with associated information about the locations and a recommended order of visiting them (Clarke and Driver, 2004). A trail makes it possible to explore adaptive characteristics common to all mobile, context-aware applications.

In Figure 4 context is associated with three types namely user, Web service, and composite service (with an emphasis on the user context throughout this paper). According to (Henricksen and Indulska, 2004) context models are classified into sensed, static, profiled (user-supplied), and derived. The U-context complies with sensed and profiled models; for instance user's location is sensed and user's interests constitute a part of his profile. The W-context complies with the sensed model; for instance a service's execution status is sensed. Finally, The C-context complies with the derived model as it relies upon the W-contexts of the component Web services to derive the context of a composite service. In the context model of (Henricksen and Indulska, 2004), user activities are in the form of a temporal fact type that covers past, present, and future activities. We are adopting a similar representation for users' activities with the various arguments that associate services with previous, current, and next periods of time. In addition, associations between users and their communication channels and devices as reported in

[2] A user profile is built-upon the long-term attributes.

[3] The 3W Composite Capabilities/Preferences Profiles group is developing standards that enable service providers to request the devices' capabilities through their profile.

(Henricksen and Indulska, 2004) are considered in the U-context.

7 Proof-of-concept development

A proof-of-concept prototype for demonstrating the feasibility of the view-based approach was developed. We use Jbuilder 9.0, which has a toolkit for building, testing, and deploying Web services, and includes as well an explorer facility for publishing and searching for Web services. For demonstration purposes, we assume that contextual information associated with user is already made available and converted into XML (Figure 9).

```
<User xml:about="Melissa">
      <!-Static user information->
      <name> Melissa </name>
      <age> 24 </age>
      <gender> F </gender>
      …
</User>
<Situation>
      <!-Current local information->
      <location> Dubai </location>
      <action> Walking </action>
      …
</Situation>
<State>
      <!- User state information>
      <cognitive> … </cognitive>
      <psychological> … </ psychological>
      …
</State>
<Topic>
      <!- Topic Description->
      <name> Computer </name>
      <level> 5 </level>
      …
</Topic>
```

Figure 9: Melissa's current context in XML

Figure 10 overviews the prototype's major functionalities: translation of composite service specification into XML, and checking of contextual information. The translation is a two-step process. The first step describes the composition itself after mapping the specification of this composition from state chart diagram into BPEL. Current implementations of BPEL do not provide the flexibility and adaptability that composite processes require during changing business rules (Rosenberg and Dustdar, 2005). Thus, BPEL processes cannot handle context changes between component Web services. We decided on automating this part of the translation process by extending BPEL. This extension consists of representing the information contained in a state chart diagram using a new element called <**transition**>. This element is added to a BPEL specification and comprises 3 attributes: *transition name*, *condition* (satisfied, unsatisfied, unknown) that describes when a Web service is available, and *color* (red, blue, green) that refers to the current state of the condition. The second step of the translation process describes the rules that apply to a BPEL specification in an XSLT template file. These rules compare the values of the contextual information to the conditions expressed in the BPEL specification, and proceed afterwards with changing the state of Web services in this specification when the comparison is valid.

With regard to the checking functionality of Figure 10, it uses an XML schema to describe the context structure and check that contextual information fits into this structure. Once contextual information parameters are fed into the XSLT template, we can first, compare the effective values of context to the extension elements of the BPEL specification and second, generate a derived specification (we recall that a derived specification is also subject to additional views). Web services that have unsatisfied conditions are removed from the BPEL specification, so that a new view over the previous specification is created. For example let us assume that a Web service has <*transition name="weather" condition="sunny" color="green"*> element. If the weather context is assessed as "**rainy**", then the state of the Web service will be switched to "**red**" standing for unavailable. Thus the Web service is discarded from the derived specification. The XSLT template uses the XPath query engine in order to locate the elements of the specification that match the contextual information it has received.

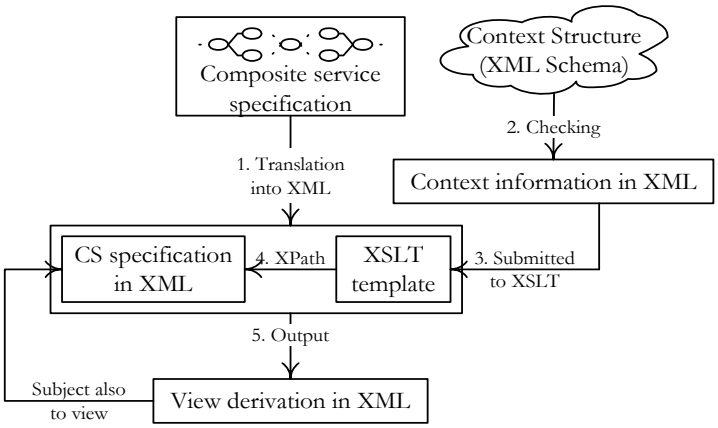

Figure 10: Operations in the view prototype

We also started the development of the various procedures (Appendix 1) that manage the dynamic nature of a derived state chart diagram. For instance, once this diagram is obtained, the current context is updated with the latest changes in the environment. The objective is to ensure that these changes are automatically reflected first, in the context and second, in the derived state chart diagram before this diagram can be subject to another view operation.

8 Conclusion

In this paper, we presented a view-based approach for tracking the execution progress of Web services. A view permits zooming into the specification that composes Web services. We have also shown that while much of the R&D on Web services to date have focused on low-level standards for publishing, discovering, and invoking Web services, it is deemed appropriate to track Web services so that user preferences are properly handled. Our contributions are summarized as follows: a clear definition of what a view is in the context of composite Web services, mechanisms for extracting and visualizing views over specifications of composite services, and mechanisms (a set of procedures) for managing the dynamic nature of these specifications in terms of adding/discarding Web services to/from specifications. A proof-of-concept prototype to demonstrate the feasibility of views was developed using various technologies such as JBuilder, XPath, and XML.

Before we highlight our future work, we suggest some features that characterize a view run over a composite service specification (adapted from (Liu and Shen, 2003)): *membership* - a derived specification, which is the outcome of running a view, is always comprised in the initial specification; *chronology Preservation* - the order of operations in a specification subject to a view is always maintained in its derived specification; and *temporal validity* - a derived specification remains valid as long as the context does not change, otherwise, a new view is run again.

Acknowledgments

The authors would like to thank Michael Mrissa for his contributions to the project described in this research paper.

REFERENCES

Allen, J. F. "Maintaining Knowledge about Temporal Interval", Communications of the ACM, Vol. 26, No. 11:832-843, 1983.

Benatallah B., M. Dumas, Q.Z. Sheng, and A. Ngu, "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services", Proceedings of The Eighteen International Conference on Data Engineering (ICDE'2002), San Jose, USA, 2002.

Berardi, D., D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "A Foundational Vision for e-Services", Proceedings of Ubiquitous Mobile Information and Collaboration Systems Workshop (UMICS'2003), Klagenfurt/Velden, Austria, 2003.

Boudriga, N. and M.S. Obaidat, "Intelligent Agents on the Web: A Review", Computing in Science Engineering, Vol. 6, No. 4:44-49, 2004.

Brézillon, P. "Focusing on Context in Human-Centered Computing", IEEE Intelligent Systems, Vol. 18, No. 3:62-66, 2003.

Bunting, D., M. C. O. Little, M. Mischkinsky, J. Newcomer, E. Webber, and K. Swenson, "Web Services Composite Application Framework (WS-CAF)", Version 1.0, 2003.

Chen, M. and M.J. Meixell, "Web Services Enabled Procurement in the Extended Enterprise: An Architectural Design and Implementation", Journal of Electronic Commerce Research, Vol. 4, No. 1:140-155, 2003.

Chen, M., A.N.K. Chen, B. Benjamin, and M. Shao, "The Implications and Impacts of Web Services to Electronic Commerce Research and Practices", Journal of Electronic Commerce Research, Vol. 4, No. 1:128-139, 2003.

Chiu, D. K. W., S. C. Cheung, S. Till, K. Karlapalem, Q. Li, and E. Kafeza, "Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment", Information Technology and Management Journal, Vol. 5, No. 3/4:221-250, 2004.

Clarke, S. and C. Driver, "Context-Aware Trails", IEEE Computer, Vol. 37, No. 8:97-99, 2004.

Curbera, F., R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The Next Step in Web Services", Communications of the ACM, Vol. 46, No. 10:29-34, 2003.

Dustdar, S. and W. Schreiner, "A Survey on Web Services Composition", International Journal on Web and Grid Services, Vol. 1, No. 1:1-30, 2005.

Fuchs, M. "Adapting Web Services in a Heterogeneous Environment", Proceedings of The Second IEEE International Conference on Web Services (ICWS'2004), San Diego, USA, 2004.

Ghedira, C., P. Marret, J. Fayn, and P. Rubel, "HANS: A Meta-Model for the Development of Adaptive User Interfaces in an Hypermedia Environment", Proceedings of The Second IEEE International Symposium on Signal Processing and Information Technology (ISSPIT'2002), Marrakech, Morocco, 191-196, 2002.

Harel, D. and A. Naamad, "The STATEMATE Semantics of Statecharts", ACM Transactions on Software Engineering and Methodology, Vol. 5, No. 4:293-333, October 1996.

Henricksen, K. and J. Indulska, "A Software Engineering Framework for Context-Aware Pervasive Computing", Proceedings of The Second IEEE International Conference on Pervasive Computing and Communications (PerCom'2004), Orlando, Florida, USA, 2004.

Kwok, S.H. and R.T. Chi, "Digital Rights Management for Mobile Commerce Using Web Services", Journal of Electronic Commerce Research, Vol. 7, No. 1:1-13, 2006.

Leymann, F. "Web Services Flow Language", IBM Corporation, 2001.

Liu, D. R. and M. Shen, "Workflow Modeling for Virtual Processes: An Order-Preserving Process-View Approach", Information Systems, Vol. 28, No. 6:505-532, 2003.

Lonsdale, P. and H. Beale, "Towards a Dynamic Process Model of Context", Proceedings of The First International Workshop on Advanced Context Modeling, Reasoning, and Management, Nottingham, England, 2004.

Ma, K. J. "Web Services: What's Real and What's Not", IEEE IT Professional, Vol. 7, No. 2:14-21, 2005.

Maamar, Z., B. Benatallah, and W. Mansoor, "Service Chart Diagrams - Description & Application", Proceedings of The Alternate Tracks of The Twelve International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.

Maamar, Z., D. Benslimane, and N. C. Narendra, "What Can Context do for Web Services?", Communications of the ACM, 2006 (forthcoming).

Maamar, Z., S. Kouadri Mostéfaoui, and Q. H. Mahmoud, "On Personalizing Web Services Using Context", International Journal of E-Business Research, Special Issue on E-Services, Vol. 1, No. 3:41-62, 2005.

Milanovic, N. and M. Malek, "Current Solutions for Web Service Composition", IEEE Internet Computing, Vol. 8, No. 6:51-59, 2004.

Nassar, M., B. Coulette, X. Crégut, S. Ebersold, and A. Kriouile, "Towards a View Based Unified Modeling Language", Proceedings of The Fifth International Conference on Enterprise Information Systems (ICEIS'2003), Angers, France, 2003.

Papadis, D. and T. Sellis, "On the Qualitative Representation of Spatial Knowledge in 2D Space", The Very Large Data Bases Journal, Vol. 3, No. 4:479-516, 1994.

Pistore, M., F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso, "Planning and Monitoring Web Service Composition", Proceedings of The Eleventh International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA'2004), Varna, Bulgaria, 2004.

Roman, G. C., C. Julien, and A. L. Murphy, "A Declarative Approach to Agent-Centered Context-Aware Computing in Ad Hoc Wireless Environments", Proceedings of The Second International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'2002), Orlando, Florida, USA, 2002.

Rosenberg, F. and S. Dustdar, "Business Rules Integration in BPEL - A Service-Oriented Approach", Proceedings of The Seventh International IEEE Conference on E-Commerce Technology (CES'2005), Munich, Germany, 2005.

Sabou, M. D. Richards, and D. Van Splunter, "An Experience Report on Using DAML-S", Proceedings of The Twelve International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.

APPENDIX 1

The followings are the algorithms that we devised for managing a derived state chart diagram dscd. This consists of adding/withdrawing transitions to/from a state chart diagram based on the content of the current context. These algorithms use two functions: ongoing_Transitions(on) returns the transitions that have on as an initial state, and all_Incomings(on,colors) checks whether the colors of all the transitions with on as an output node, are part of the given set colors.

```
Proc Red_To_Blue(t)
  Input: t Red Transition
  Begin
  // Executed when the unsatisfied transition t becomes satisfied
  t.Color ← Blue // T moves to Bleu
  in ← InitialState(t)
  on ← Out_State(t)
    // Check whether the new satisfied transition t can be added to the dscd diagram and then propagate
    this change
  if in ∈ {AN} and on ∉ {AN} then
    AN ← AN ∪ {on }
    t.Color ← Green
    for t' ∈ Ongoing_Transitions (on) do
      If Color (t') == Blue then Blue_To_Green(t')
      end if
    end for
  end if
  End

Proc Blue_To_Red(t)
  Input: t Blue Transition
  Begin
  // Executed when the satisfied transition t becomes unsatisfied
  t.Color ← Red
  End

Proc Blue_To_Green(t)
  Input: t Blue Transition
  Begin
    // Executed when the satisfied transition t is to be added to the dscd diagram.
  t.Color ← Green
  // Add to dscd diagram all the satisfied transitions which have output state of t as an initial state
  on ← Out_State(t)
  if on ∉ {AN} then
    AN ← AN ∪ {on }
    for t' ∈ Ongoing_Transitions(on) do
      if Color (t') == Blue then Blue _To_ Green (t')
      end if
    end for
  end if
  End

Proc Green_To_Red(t)
  Input: t Green Transition
  Begin
  // Executed when the satisfied transition t becomes unsatisfied
  t.Color ← Red
  on ← Out_State(t)
```

```
        // Withdraw from dscd diagram some green transitions with on as an output state.
      if all_Incomings(on,{red, blue})==true then
        AN ← AN – {on}.
        for t' ∈ Ongoing_Transitions(on) do
          if Color (t') == Green then Green_To_Blue(t')
          end if
        end for
      end if
8.1 End

    Proc Green_To_Blue(t)
      Input: t Green Transition
8.2 Begin
      // Executed when the green satisfied transition t changes to blue
      t.Color ← Blue
      on ← Out_State(t)
      // Withdraw from dscd diagram some green transitions with on as an output state
      if all_Incomings(on,{red, Blue})==true then
        AN ← AN – {on}
        for t' ∈ Ongoing_Transitions(on) do
          if Color (t') == Green then Green_To_Blue(t') end if
        end for
      end if
      End

    Proc Black_To_Blue(t)
      Input: t Black Transition
8.3 Begin
      // Executed when the unknown transition (neither satisfied nor unsatisfied) t becomes satisfied.
      t.Color ← Blue
      in ← InitialState(t)
      on ← OUT_State(t)
        // Check whether the new satisfied transition t can be added to the dscd diagram and then propagate
        this change.
      if in ∈ {AN} and on ∉ {AN} then
        AN ← AN ∪ {on}
        t.Color ← Green
        for t' ∈ Ongoing_transitions (on) do
          if color (t') == Blue then Blue_To_Green(t')
          end if
        end for
      end if
  9 End

    Proc Black_To_ Red (t)
      Input: t Black Transition
9.1 Begin
      // Executed when the unknown transition (neither satisfied nor unsatisfied) t becomes unsatisfied.
      in ← InitialState(t)
      if in ∈ {AN} then
        Green_To_Red(t)
        Else Blue_To_ Red(t)
      end if
      End
```