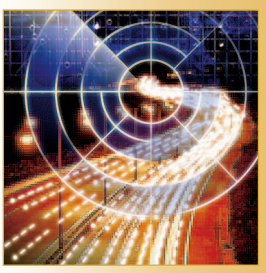
Editors: Michael Huhns • *huhns@sc.edu*
Munindar P. Singh • *singh@ncsu.edu*



Views in Composite Web Services

The authors present a view-based approach for tracking personalized Web services. To guarantee proper handling of user preferences during Web service execution, a view can zoom into the specification that composes the Web services. As time advances, the location changes, or constraints on the environment are satisfied, the deployment of a view over a specification similarly progresses, mirroring the context's dynamic nature.

Zakaria Maamar
Zayed University

Djamal Benslimane, Chirine Ghedira, and Michael Mrissa
Claude Bernard Lyon I University

Composite Web services — those that describe component execution order as well as corrective actions to take in case of exceptions — provide a new way to implement business processes that could cross organizational boundaries.

The preferences used for personalizing a Web service depend heavily on the environment in which the service operates. The ability to sense, gather, and refine an environment's features, for example, helps us define the service's *context*,[1] which, in turn, helps us adjust the service's specification.

In context-aware computing, software applications can detect and respond to changes in their environments.[2] A *view* is a dynamic snapshot of the environmental changes that occur in a composite service's entire specification according to a certain context. Views highlight what was expected to happen versus what actually is happening, and they offer some important benefits:

- providing flexible security by hiding everything in the specification from the user except those parts subject to adjustment;
- identifying component Web services from user context; and
- facilitating multiple views of the specification at different levels of granularity for different users.

We propose using views as a means of both tracking the execution of personalized Web services and deploying corrective measures when services don't comply with users' personalization requirements.

Example Scenario

Our running scenario focuses on Melissa, a tourist in Dubai. After checking in at her hotel, Melissa browses some Web sites recommended by the Dubai tourism authorities in their brochures. The top-ranked Web site offers different services that can be composed according to different execution chronologies.

 1089-7801/05/$20.00 © 2005 IEEE Published by the IEEE Computer Society

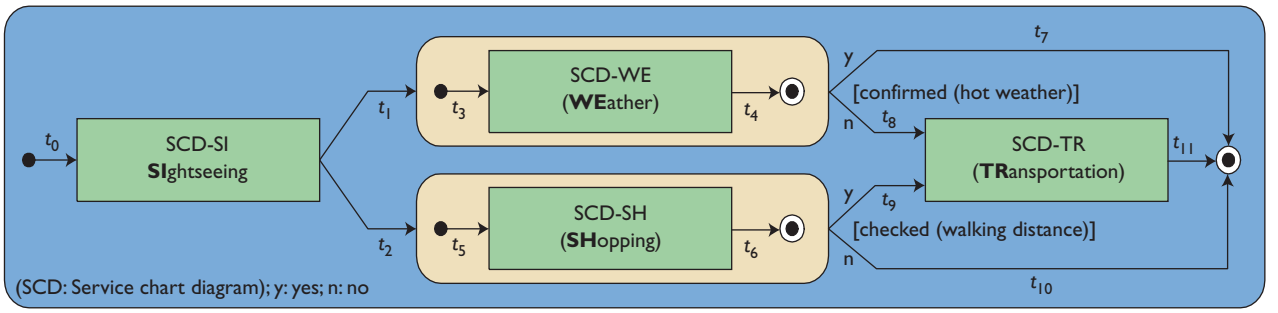


Figure 1. Composite service specification. For Melissa's trip, the component services are sightseeing (SI), weather (WE), shopping (SH), and transportation (TR).

Melissa plans to visit outdoor places in the morning and go shopping in the afternoon. The first part of this plan depends on weather forecasts: hot weather often forces tour operators and other service providers to cancel outdoor activities. Melissa is initially prompted to select activities based on her interests, indicate pick-up and drop-off places and times for sightseeing and shopping, and request a guide for specific visits.

The selected Web site then submits her preferences to the appropriate Web services providers for processing. The sightseeing service obtains forecasts from the weather service; if there is no warning of hot weather, this service prepares a schedule for Melissa, verifying if the places she wants to see are open to the public on her desired days and arranging rides and a guide. The transportation service identifies the type of vehicle she'll be using and whether she'll have to share a ride with other tourists. If the weather is too hot, the sightseeing service might suggest indoor places to visit, such as museums. Similar considerations apply for shopping, which involves checking for any promotions or sales occurring in the malls that Melissa selected. The transportation service uses the distance among destinations to help coordinate the timings among all the activities. Finally, the transportation service conveys Melissa's travel plan — which specifies the duration and order of her selected activities — to her PDA.

The day after her arrival, Melissa takes a ride to a historical site, but because of an unexpected traffic jam, she's delayed. The agent running on her PDA detects that she isn't in the expected location according to her travel plan, so it informs the sightseeing and transportation services so that they can take corrective measures — for example, informing the guide waiting for her about the delay and adjusting her shopping trip.

Specification of Composite Web Services

A *service chart diagram* specifies a composite service's components.[3] It enhances a state chart diagram by emphasizing the context surrounding a service's execution rather than just the states during service execution. Five perspectives enhance a service's states — the state itself, flow, business, information, and performance — but for personalization, users indicate when and where they want services performed and results delivered. To incorporate these preferences, we anchor two extra perspectives — location and time — to the service chart diagram.[4,5]

Several component services can form a composite service, so we specify the process model underlying the composite service as a state chart diagram with associated states. Figure 1 illustrates the composite service specification for Melissa's scenario. The services are connected through transitions, some of which are constrained via Boolean expressions (for example, `[confirmed(hotweather)]`).

Understanding Views

Our view-based approach isn't specific to service chart diagrams: it's applicable to any composition language. We use service charts for illustration purposes only.

View Metamodel

Figure 2 shows the view metamodel for tracking personalized services. This metamodel revolves around

- context, service, and composite service, the

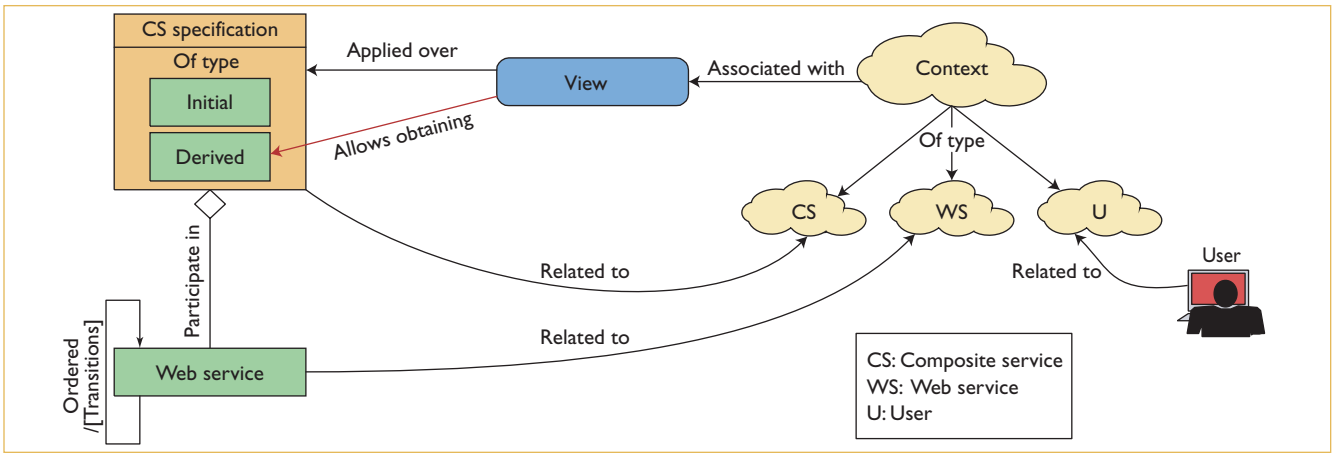


Figure 2. The view metamodel. The rounded rectangle in the center represents the view, and regular rectangles represent individual and composite service concepts.

building blocks of any context-aware, service-based system,[6] and
- the mechanisms of running context-based requests over composite service specifications.

The view brings these mechanisms to life. Figure 2 uses a rounded rectangle to represent a view to differentiate it from individual and composite service concepts, which are represented with regular rectangles.

In Figure 2, we decompose context into three types: user-linked (U), Web-services-linked (WS), and composite-service-linked (CS). Figure 2 uses related-to edges to highlight the connection between the three. U-context tracks users in terms of current location and current activities, WS-context refers to a Web service's current capabilities and ongoing participation in concurrent compositions, and CS-context oversees the execution status of a composite service's components.

A composite service specification is either initial or derived. A designer creates an initial specification, which includes details such as the component services' chronology and the dependencies between them. Figure 1 is an initial composite service specification. The designer obtains a derived specification after running a view over a specification, which itself is initial or derived. In Figure 2, we highlight the connection between view and composite service specification with two different edges: "applied over" (black) and "allows obtaining" (red). The aggregation of services into composite services occurs via the "participate in" edge in the bottom left-hand section. This participation complies with a specific execution chronology that corresponds to the `Ordered/[Transitions]` edge. `[Transitions]` represents the conditions of connecting Web services together.

The view metamodel's dynamic aspect involves two steps. The first consists of checking the constrained transitions before running a view over a specification (for example, `[checked(walkingdistance)]`). By checking transitions, we limit the services included in the specification to those subject to a view operation. Once the features in the current context satisfy these constraints, we move to the second step: identifying the parameters included in the extraction of a view from a composite specification. To perform this extraction, we use two parameters: execution time and execution location.

Views Formalized

As we've just seen, a view depends on the user preferences and constraints that regulate the transitions between component services. We've also learned that execution time and location are examples of preferences, and weather forecasts and walking distance are examples of constraints. Here, we present a formal specification of the view concept and an application of this specification to Melissa's scenario.

First, we define the state chart diagram. A state chart diagram *scd* for a composite service is a triple $<S, T, Tc>$, where $S = \{s_1, s_2, ...\}$ is the set of temporal and localized service chart diagrams for the component services; $T = \{t_1, t_2, ...\}$ is the set of unconstrained transitions; and $Tc = \{tc_1, tc_2, ...\}$ is the set of constrained transitions whose constraints are suitable Boolean expressions.

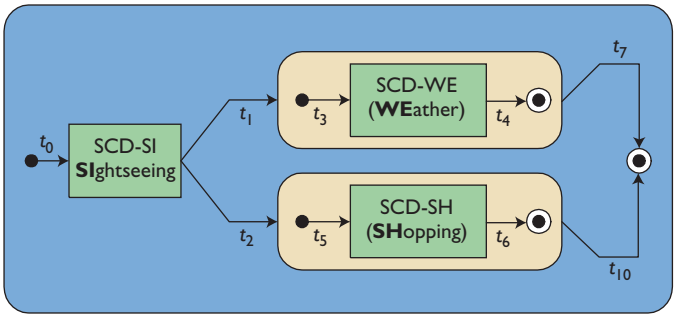


Figure 3. Derived state chart diagram. The first step checks the constrained transitions before running any view over a specification. The second step identifies the time and location parameters used to extract a view from the specification.

Next, we define the context. A context *cont* aggregates user, Web service, and composite service contexts. This article considers only the U-context, which involves user-specific features such as role, location, and expertise level.

Finally, we use a derived state chart diagram *dscd* to extract a view from *scd* according to a given *cont*. Given $scd = <S, T, Tc>$, we can extract a triple *dscd* given by $View(scd, cont) = <S', T', Tc'>$ from *scd* only if the following hold:

- $S' \subseteq S$. This means that the derived specification can't accept any additional services, but can exclude existing elements such as states and transitions. In particular, if the constraints on an incoming transition of *scd* aren't satisfied in *cont*, this service chart diagram will be excluded from the derived state chart diagram.
- $T' = \{t' \mid \text{either } (t' \in T \text{ and } InitialState(t') \in S') \text{ or } (\exists tc \in Tc \mid t' = FullInstantiation(tc, cont) \text{ and } InitialState(t') \in S')\}$. The function *InitialState* determines a transition's initial state. The function *FullInstantiation* returns an unconstrained transition when the constraint on this transition is satisfied in *cont*. Therefore, we obtain the unconstrained transitions in T' of *dscd* either from the unconstrained transitions of *scd* for which the initial state chart diagram belongs to S' or the constrained transitions of *scd* satisfied in *cont*.
- $Tc' = \{tc' \mid (tc' \in Tc \text{ and } UnSatisfied(tc', cont) = false \text{ and } InitialState(tc') \in S')\}$. Here, Tc' is the set of constrained transitions tc' that are unknown (that is, neither satisfied nor unsatisfied) in *cont*. Moreover, the initial state of these constrained transitions is an element of S'. The function *UnSatisfied* checks whether a transition is satisfied in a given context.

Figure 1 represents the state chart diagram $scd_{Melissa}$ that implements Melissa's scenario. The diagram is a triple $<S, T, Tc>$, where $S = \{scd\text{-}si, scd\text{-}we, scd\text{-}sh, scd\text{-}tr\}$, $T = \{t_0, t_1, t_2, t_{11}\}$, and $Tc = \{t_7, t_8, t_9, t_{10}\}$. Let's assume that Melissa's context returns details on weather conditions and walking distance between malls: $cont = ((confirmed(hotweather) = yes$ and $checked(walkingdistance) = yes)$. Figure 3 corresponds to *dscd*, extracted from $scd_{Melissa}$ for *cont*; *dscd* is defined by a triple $<S', T', Tc'>$ where $S' = \{scd\text{-}si, scd\text{-}we, scd\text{-}sh\}$, $T' = \{t_0, t_1, t_2, t_7, t_{10}\}$, and $Tc' = \emptyset$.

The transitions t_7 and t_{10} constrained in *scd* turn out to be unconstrained transitions in *dscd*. Their respective constraints are satisfied in the current context — namely, $cont = (confirmed(hotweather) = yes, checked(walkingdistance) = yes)$. Not satisfied in context *cont* are the unconstrained transitions t_8 and t_9, which aren't included in *dscd*. In general, a derived state chart diagram continually evolves with the dynamic user context. Constrained transitions, for example, become unconstrained when more information about the user's context is available. In Melissa's case, the derived state chart diagram we ultimately get is qualified as final because all of its transitions are unconstrained. By using the service chart diagram's time and location perspectives, the current time and location values allow the designer to detect the services being executing.

View Automation

Last fall in Lyon, we developed a prototype of our view-based approach for tracking personalized Web services. Figure 4 overviews the prototype's major functionalities: translating the composite service specification into XML and checking contextual information.

The first step in translation is to describe the composition itself after mapping its specification from a state chart diagram into the Business Process Execution Language. Because BPEL doesn't provide the flexibility and adaptability that composite processes require during changes to business rules,[7] it can't handle context changes among component Web services. We worked toward automating this part of the process by extending BPEL; our extension consisted of representing the information contained in a state chart

diagram with a new element called <*transition*>, which comprises three attributes: transition name; condition (satisfied, unsatisfied, or unknown), which describes when a Web service is available; and color (red, blue, or green), which represents the condition's current state.

The second step of the translation process is to describe the rules that apply to a BPEL specification in an Extensible Stylesheet Language Transformations (XSLT) template. These rules first compare the values of the contextual information to the conditions expressed in the BPEL specification and then change the state of the Web services in this specification accordingly.

For the checking functionality, we use 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 first compare the context's effective values to the BPEL specification's extension elements and then generate a derived specification (which can be subject to additional views). This comparison removes Web services with unsatisfied conditions from the BPEL specification, and creates a new view over the previous specification. Let's assume, for example, that a Web service has an element `<transition name = "weather" condition = "sunny" color = "green">`. If the weather context is assessed as "rainy," the Web service's state switches to "red," which stands for unavailable — this discards the Web service from the derived specification. The XSLT template uses the XPath query engine to locate elements of the specification that match any contextual information it receives.

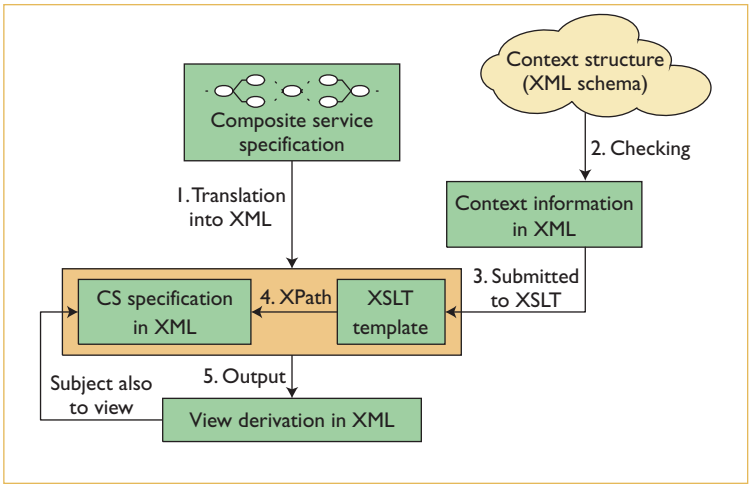


Figure 4. View prototype. It requires two steps: composite service specification translation into XML and contextual information verification.

Melissa's scenario hints at some of the challenges involved in tracking personalized Web services, but bigger obstacles include the fact that current services often can't be embedded with context-aware mechanisms, existing approaches facilitate choreography but neglect the impact of context, and guidelines for tracking personalized Web services are practically nonexistent. The scenario also raises interesting questions: How do designers detect a problem during specification execution? Which services must be checked out in case of problems? Is the context sufficient for detecting potential problems? These questions highlight the importance of making Web services aware of their surrounding environments prior to any engagement in composition.

The capacity to reuse derived specifications to be subject to views is an argument in favor of adopting views as a mechanism for tracking Web services. If the environment changes (in terms of context content/time period/location) after generating a derived state chart diagram, this diagram must be reviewed before it can be subject to another view operation. This could result in adding or discarding transitions and states to or from this diagram, which means we need to generate a new diagram.

References

1. P. Brézillon, "Focusing on Context in Human-Centered Computing," *IEEE Intelligent Systems*, vol. 18, no. 3, 2003, pp. 62–66.
2. G.C. Roman, C. Julien, and A.L. Murphy, "A Declarative Approach to Agent-Centered Context-Aware Computing in Ad Hoc Wireless Environments," *Proc. 2nd Int'l Workshop Software Eng. for Large-Scale Multi-Agent Systems* (SELMAS), Springer, 2002, pp. 94–109.
3. Z. Maamar, B. Benatallah, and W. Mansoor, "Service Chart Diagrams: Description and Application," *Proc. Alternate Tracks 12th Int'l World Wide Web Conf.* (WWW), ACM Press, 2003; http://www2003.org/cdrom/papers/alternate/P043/p43-maamar.pdf .
4. J.F. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. ACM*, vol. 26, no. 11, 1983, pp. 832–843.
5. D. Papadis and T. Sellis, "On the Qualitative Representation of Spatial Knowledge in 2D Space," *Very Large Databases J.*, vol. 3, no. 4, 1994, pp. 479–516.

6. Z. Maamar, D. Benslimane, and N.C. Narendra, "What Can Context Do for Web Services?," to be published in *Comm. ACM*, 2005.
7. F. Rosenberg and S. Dustdar, "Business Rules Integration in BPEL: A Service-Oriented Approach," *Proc. 7th Int'l IEEE Conf. E-Commerce Technology* (CES), IEEE Press, 2005, pp. 476–479.

Zakaria Maamar is an associate professor of computer science at Zayed University, Dubai, United Arab Emirates. His research interests include Web services, software agents, and context-aware computing. Maamar has a PhD in computer science from Laval University. Contact him at zakaria.maamar@zu.ac.ae.

Djamal Benslimane is a full professor of computer science at Claude Bernard Lyon 1 University and a member of the Laboratoire d'Informatique en Images et Systèmes d'Information, Central National de la Recherche Scientifique (LIRIS-CNRS), both in Lyon, France. His research interests include interoperability, Web services, and ontologies. Benslimane has a PhD in computer science from Blaise Pascal University. Contact him at djamal.benslimane@liris.cnrs.fr.

Chirine Ghedira is an associate professor of computer science at Claude Bernard Lyon 1 University and a member of LIRIS-CNRS, both in Lyon, France. Her research interests include Web services and context-aware computing. Ghedira has a PhD in computer science from the National Institute for Applied Sciences (INSA). Contact her at chirine.ghedira@liris.cnrs.fr.

Michael Mrissa is a PhD candidate in computer science at Claude Bernard Lyon 1 University and a member of LIRIS-CNRS, both in Lyon, France. His research interests include Web services and peer-to-peer networks. Contact him at michael.mrissa@liris.cnrs.fr.