On Tracking Personalized Web Services Using Views

Zakaria Maamar
Zayed University, U.A.E
zakaria.maamar@zu.ac.ae

Djamal Benslimane, Chirine Ghedira, and Michael Mrissa
Université Claude Bernard Lyon 1, France
firstname.lastname@liris.cnrs.fr

Abstract

This paper presents a view-based approach for tracking personalized Web services. Web services are subject to personalization when there is a need to accommodate user preferences during these Web services performance and delivery of this performance's outcome. Preferences are of multiple types such as when the execution of a Web service should be initiated and where the outcome of this execution should be returned. To guarantee that these preferences are handled during Web services execution, a view offers the opportunity of zooming into the specification that composes Web services. As time advances, location changes, or constraint on the environment becomes satisfied, the deployment of a view over a specification progresses, featuring the dynamic nature of tracking personalized Web services.
Keywords. *Web services, Personalization, View, Tracking.*

1 Introduction

Web services are among the technologies that help organizations connect their business processes to other peers' processes. Web services-based Business-to-Business scenarios show the capacity of Web services to be composed into *composite services*. Composition primarily addresses the situation of a user's request that cannot be satisfied by any available Web service (called service in the rest of this paper), whereas a composite service obtained by combining available Web services might be used [2].

A composite service is always associated with a *specification*, which describes among others the list of component Web services, the execution order of these component Web services according to various dependencies, and the corrective strategies for exception handling. People (e.g., researchers, IT programmers) have adopted different languages for specifying composition of Web services such as BPEL and WSFL. The aim of these languages is to provide a high-level description of the composition process far away from any implementation concerns.

The integration of user preferences into a Web service composition permits Web services *personalization*. Some users would like receiving answers to their personal requests directly submitted to their personal email instead of office's email. Personalization depends on the features of the environment in which the Web services are to be executed. These features can be about users, computing resources, physical locations, etc. Sensing, gathering, and refining the features and changes of an environment allows defining what is known as *context*. Context is the information that characterizes the interactions between humans, applications, and the surrounding environment [4].

In this paper, the primary use of context is to adjust the specification of a composite service according to the features of the environment. For instance, knowing that a user's laptop is currently facing some low performance problems should trigger the request of postponing the outcome delivery of the service this user has initiated earlier. Context-aware computing refers to the ability of a software application to detect and respond to changes in its environment [10]. To be aware of which part of the specification of the composite service has to be *adjusted* because of the changes of 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 between the "expected" and "happening" enables detecting the discrepancies so that corrective measures are taken. We define a view as a dynamic snapshot over a specification of a composite service according to a certain context. 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., whereas **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 addition it will be shown in this paper that besides the view that implements the context of user over the specification of a composite service, two additional views might be considered. The first view implements the context of a Web service, and the second view implements the context of a composite service. Embedding services whether Web or composite with context-awareness mechanisms has several advantages as Maamar et al. report in [7].

While much of the R&D on Web services to date have

focused on low-level standards for publishing, discovering, and invoking Web services, respectively, it is deemed appropriate tracking personalized Web services so that user preferences are properly handled. However, *very little* has been achieved to date regarding this tracking due to several obstacles current Web services do not act as active components that can be embedded with context-awareness mechanisms, existing specification approaches for Web services composition typically facilitate choreography only while neglecting contexts and their impact on this choreography, and lack of guidelines supporting the operations of Web services personalization and tracking. In this paper we propose the use of views as a support means for first, tracking the execution progress of personalized Web services and second, deploying corrective measures in case of non-compliance with the personalization requirements as users indicate for these Web services. Our contributions are summarized as follows: 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; and mechanisms for extracting and visualizing views over specifications of composite services.

Section 2 provides a scenario for motivating the adoption of views. Section 3 discusses the rationale of views. Section 4 details the view-based tracking of personalized Web services. Section 5 presents the mathematical model of a view. The implementation status of a view-based tracking is discussed in Section 6. Prior to concluding in Section 8, we present some related work in Section 7.

2 Motivating scenario

Our motivating scenario concerns Melissa, a tourist who just landed in Dubai. Melissa plans using her PDA as a tourist *mobilebook* instead of carrying brochures and booklets. Upon arrival to the airport she decides on downloading some applications into her PDA as they are 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 composed according to a certain pattern, and she decides on doing so. Melissa's plans are to visit outdoor places in the morning and go for 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. Melissa is prompted among others to select some outdoor places, to express her need for a guide during the visits, etc.

Once Melissa's preferences are set, they are submitted to appropriate Web services so that their processing can now be initiated. With regard to the first activity, `sightseeing` Web service checks with `weather` Web service the forecasts for the five coming days. If there is no warning of hot weather, the scheduling of the places to visit begins by ensuring that these places are open for public on these days, and transportation and guide are arranged. The logistics of Melissa's rides is affected to `transportation` Web service, which identifies for instance the type of vehicle and the possibility that Melissa commutes with other tourists heading towards the same places. In case of hot-weather warning, `sightseeing` Web service can suggest places (e.g., museums) where indoor activities can take place. The same description applies to the shopping activity, which consists of checking out the running promotions in the malls that Melissa has selected. If Melissa has included several malls during her shopping plan, the distance between them is considered so that appropriate transportation is scheduled, too. It should be noted that shopping's starting time is coordinated with sightseeing's finishing time. `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 returned 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 such as shopping. 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 [3]) 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 according to the previously-defined specification, her PDA takes actions by notifying `sightseeing` and `transportation` Web services so that corrective measures are performed, e.g., informing the guide waiting for Melissa about the delay.

3 Why using views?

Various reasons motivate integrating views into the tracking of personalized 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 highlighted. Second, the view mechanism allows identifying the component Web services of a specification in a way that is customized to the context of user. This customization can also include the contexts of Web services and composite services. Third, because the same specification of a composite service is offered to several users for triggering, multiple views over this specification are 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 adopt.

4 View's foundations

Before we discuss the foundations on which the view concept is built upon, we first overview state and service chart diagrams. A specification of Melissa scenario is also included in this overview. In our research on views we adopt these diagrams for the specification of composite services.

4.1 State/Service chart diagram

A state chart diagram is a graphical representation of a state machine that visualizes how and under what circumstances a modelled element changes its states. 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 modelling and specifying the component Web services of a composite service [6]. It enhances a state chart diagram, putting the emphasis on the context surrounding the execution of a Web service rather than only the states that a Web service takes (Fig. 1). To this end, the states of a service are wrapped into five perspectives namely: *state*, *flow*, *business*, *information*, and *performance*.

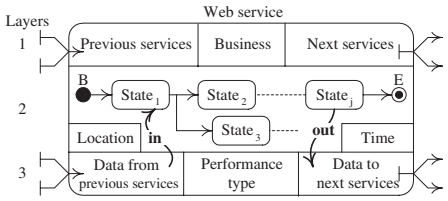

Figure 1. Service chart diagram of a Web service

When it comes to personalizing Web services, users indicate when and where they would like having these Web services performed according to particular periods of time and particular locations (additional types of preferences could be added). Users may also indicate when and where they would like having the outcome of performing the Web services delivered. To illustrate user preferences, two extra perspectives denoted by *location* and *time* are anchored to a service chart diagram (Fig. 1)[1]. Location identifies particular places (e.g., classroom), and time identifies particular moments (e.g., at 8am).

Because a composite service is made up of several component services, the process model underlying the composite service is specified as a state chart diagram. In this diagram, states are associated with the service chart diagrams of the component services (Fig. 1), and transitions are labelled with events, conditions, and variable assignment operations. Fig. 2 is the specification of the composite service of Melissa scenario. Each component Web service has a service chart diagram: sightseeing (`SI`), weather (`WE`), shopping (`SH`), and transportation (`TR`). These diagrams are connected through transitions; some of these transitions have constraints to satisfy (e.g., [confirmed(hot weather)]).

[1]The semantics of time/location perspective is based on [1]/[9].

4.2 The view meta-model

Fig. 3 is the view meta-model for tracking personalized Web services. This meta-model revolves around two types of concept. The first type of concept identifies the building blocks of a context-aware and Web services-based system. These blocks are 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 materializes this mechanism. A view is represented with a rounded rectangle in Fig. 3, so that it is differentiated from Web service and composite service concepts represented with regular rectangles.

Context is decomposed into three types: user which we refer to as $\mathcal{U}$-context, Web service which we refer to as $\mathcal{W}$-context, and composite service which we refer to as $\mathcal{C}$-context. While the role and structure of each context are not within this paper's scope, we refer readers to [7] for more details. In addition we recall that we only focus on views associated with $\mathcal{U}$-contexts of users. The connection (context, user/Web service/composite service) is highlighted in Fig. 3 with *related to* lines. $\mathcal{U}$-context enables tracking user in terms of current location, current activities, etc. $\mathcal{W}$-context refers to the current capabilities and ongoing participations of a Web service. Finally, $\mathcal{C}$-context informs about the execution status of the multiple component Web services of a composite service.

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, in which he describes for instance the chronology of the component Web services of a composite service and the types of dependency between them. Fig. 2 is a composite service specification of type initial. A derived specification is obtained after running a view over a specification that might be either of type initial or derived. To illustrate these two types of specification, the connection (view, composite service specification) is highlighted in Fig. 3 with two different lines: *applied over* as regular line and *allows obtaining* as dashed line. Thus 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 already motivated in Section 3.

Last but not least, the aggregation of Web services into composite services is highlighted in Fig. 3 with *participate in* line. This participation complies with a specific execution chronology that corresponds to *or-*

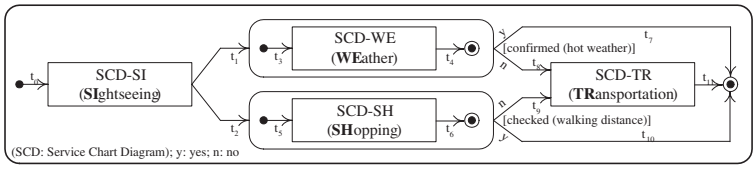

Figure 2. Specification of the composite service of Melissa scenario

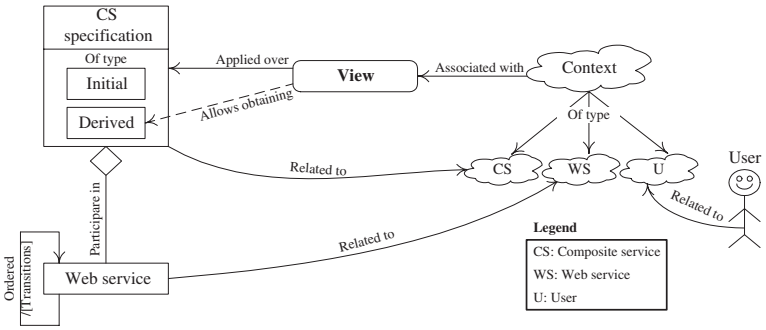

Figure 3. Representation of the view meta-model

dered/[transitions] line. *[Transitions]* represents the order of connecting Web services together.

The dynamic aspect of the view meta-model has two steps. The first step consists of checking the transitions that have constraints (e.g., [checked(walking distance)] in Fig. 2) before any view is applied over a specification. Transitions limit the number of Web services of a composite service that need to be triggered. Once these constraints are satisfied because of the features of the current context, the second step consists of identifying the parameters that will be included in the extraction of a view from a composite service specification. We recall that we have identified two parameters: execution time and execution location.

5 View's mathematical model

We illustrated how a view is dependent on the preferences of users and constraints that regulate the transitions between Web services. Execution time and execution location are examples of preferences, and weather forecasts is an example of constraint. This section provides a formal specification of the view concept and proceeds next with an application of this formal specification by using Melissa scenario.

5.1 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. $SCD = ISCD \bigcup DSCD$. A state chart diagram $scd \in SCD$ is a triple $scd =< S, T, Tc >$ where:*

• *$S = \{s_1, s_2, \cdots\}$ is the set of temporal and localized service chart diagrams (Fig. 1). $\forall s \in S, s = f_s(t, l)$ where t and l represent time and location parameters respectively.*

• *$T = \{t_1, t_2, \cdots\}$ is the set of unconstrained transitions.*

• *$Tc = \{tc_1, tc_2, \cdots\}$ 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 $F(\mathcal{U}, \mathcal{W}, \mathcal{C})$ where $\mathcal{U}$, $\mathcal{W}$, and $\mathcal{C}$ represent $\mathcal{U}$ser, $\mathcal{W}$eb service, and $\mathcal{C}$omposite service contexts, respectively. As mentioned earlier, only $\mathcal{U}$-context is used for extracting derived state chart diagrams; $\mathcal{W}$- and $\mathcal{C}$-contexts are discarded. $\mathcal{U}$-context is defined by the function $f_{\mathcal{U}}(arg_1, arg_2, \cdots)$, where arg_i is an user parameter such as preferences, status, location, etc.*

We now formally describe through the concept of derived state chart diagram, how a view is extracted from a given scd according to a given context $cont$, a given time period $time$, and a given location loc.

Definition 3 (**Derived state chart diagram**). *Let scd, $cont$, $time$ and loc be a state chart diagram (initial or derived), a context, a time period, and a location, respectively. A derived state chart diagram $dscd \in DSCD$ is a tuple $dscd : View(scd, cont, time, loc) =< S', T', Tc' >$ 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 namely states (i.e., illustrated with service chart diagrams in Fig. 1) 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 is not satisfied in a certain context, then this service chart diagram will be excluded from the derived state chart diagram.

- $T' = \{t' \mid$ ***either*** $t' \in T \wedge InitialState(t') \in S'$, ***or*** $\exists\ tc \in Tc \mid t' = FullInstanciation(tc, cont, time, loc) \wedge InitialState(t') \in S'\}$. *InitialState(t') is a function that determines the initial state of a transition t'. FullInstanciation$(tc, cont, time, loc)$ is a function that returns an unconstrained transition t' because the constraint on this transition, previously tc, is satisfied in the current context $cont$, the given time-period $time$, and the given location loc. Therefore, the unconstrained transitions of a derived state chart diagram $sscd$ 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 which are satisfied in the current context $cont$, current time-period $time$, and current location loc.*
- $Tc' = \{tc' \mid tc' \in Tc \wedge UnSatisfied(tc', cont, time, loc) = false \wedge InitialState(tc') \in S'\}$. *$Tc'$ is defined as the set of constrained transitions, which are satisfied in the current context, time period, and location, and for which the initial state chart diagram is an element of S'. $UnSatisfied(tc', cont, time, loc)$ checks whether transition tc' is satisfied in the current context $cont$, current time-period $time$, and current location loc.*

5.2 Application to Melissa scenario

Fig. 2 represents $scd_{Melissa}$, the state chart diagram that implements 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: $cont = (confirmed(hot\ weather) = yes \wedge checked(walkingdistance) = yes)$. Fig. 4 corresponds to the derived state chart diagram $dscd_{Melissa}$ that is extracted from $scd_{Melissa}$ for this given context $cont$. $dscd_{Melissa}$ 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 $Tc' = \emptyset$.

It should be noted that the constrained transitions t_7 and t_{10} in $scd_{Melissa}$ turn out unconstrained transitions in $dscd_{Melissa}$. Their respective constraints are satisfied in the current context namely $cont = (confirmed(hot\ weather) = yes \wedge checked(walkingdistance) = yes)$. Being not satisfied in the current context $cont$ are the unconstrained transitions t_8 and t_9; they are excluded in $dscd_{Melissa}$. Generally, a derived state chart diagram is in a constant evolution along with the dynamic nature featuring the user context. For instance constrained transitions become unconstrained when more information about the user context are available. With regard to Melissa, the obtained derived-state chart diagram is qualified as final since all its transitions are unconstrained. The current time and location values allow detecting the Web services that are under execution. This is done by using time and location perspectives of the service chart diagram.

6 Implementation status

A proof-of-concept prototype for demonstrating the feasibility of the view-based approach for tracking personalized Web services is under development. We are adopting Borland JBuilder Enterprise Edition 9.0[2]. JBuilder 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 prototyping purposes, we assume that contextual information associated with user is already made available and converted into XML.

Two major functionalities are integrating into the prototype: translation of composite service specification into XML, and checking of contextual information. The translation, which takes as input the specification of a composite service as a state chart diagram and outputs XML code, is a two-step process. The first step describes the composition itself in XML, whereas the second step describes the rules that apply to the composition specification in an XSLT template file. We are working towards automating the translation process. With regard to the checking functionality, it uses an XML schema for describing contextual information structure and checking that this information fits into the structure model. By introducing the contextual information parameters into the XSLT template, we derive the composite service specification and build a derived state chart diagram. The XSLT template uses the XPath query engine in order to locate the elements of the specification that match the contextual information it receives.

7 Related work

The notion of view is adopted by Roman et al. [10], who suggested a declarative approach to agent-centered context-aware computing in ad-hoc wireless environments. While we consider a view as a concrete implementation of context over a composite service specification, Roman et al. structure context in terms of fine-grained units referred to as views. In that case a view is a projection of the maximal context together with an interpretation that defines the

[2]http://www.borland.com/jbuilder/enterprise/index.html.

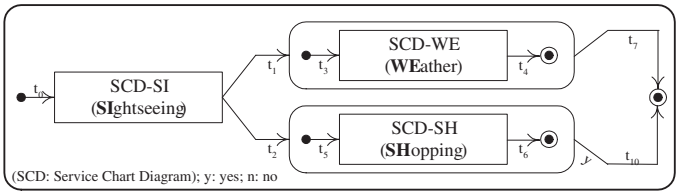

Figure 4. Sample of a derived state chart diagram

rules of engagements between a software agent and a particular view. The maximal context includes information that all hosts in the network store and the context information (e.g., location, time) 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 [8]. 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 profile, a view-extension relationship is set between a view and an extension of the default view. Several similarities are identified between our work and Nassar et al.'s work. Independently of the concept (class or state chart) on which the extraction of a view executes over, the base class corresponds to the initial specification of a composite service, and the viewpoint on the class base corresponds to a derived specification of a composite service. However a major difference between both works resides in the static *vs.* dynamic nature that features obtaining a view. Nassar et al.'s view depends on user profile that is to a certain extent static (i.e., content does not change frequently). Our view approach depends on context that is dynamic by nature (i.e., what is going on over time).

In the context model of [5], 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 [5] are considered in the $\mathcal{U}$-context.

8 Conclusion

In this paper we presented a view-based approach for tracking personalized Web services. Web services are subject to personalization when there is a need of accommodating user preferences during these Web services performance and outcome delivery of this performance. To guarantee that user preferences are properly handled during Web services execution, a view has offered the opportunity of zooming into the specification that composes Web services. As time advances, location changes, or constraint becomes satisfied, the deployment of a view over a specification progresses. Indeed, a view contains all the elements currently present within an ongoing composite service specification that are relevant to a particular context of user such as time and location.

References

[1] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11), November 1983.

[2] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. A Foundational Vision for e-Services. In *Proc. of Ubiquitous Mobile Information and Collaboration Systems Work. (UMICS'2003) held in conjunction with The 15th Int. Conf. On Advanced Information Systems Engineering (CAiSE'2003)*, Klagenfurt/Velden, Austria, 2003.

[3] N. Boudriga and M. Obaidat. Intelligent Agents on the Web: A Review. *Computing in Science Engineering*, 6(4), July-August 2004.

[4] P. Brézillon. Focusing on Context in Human-Centered Computing. *IEEE Intelligent Systems*, 18(3), May/June 2003.

[5] K. Henricksen and J. Indulska. A Software Engineering Framework for Context-Aware Pervasive Computing. In *Proc. of The Second IEEE Int. Conf. on Pervasive Computing and Communications (PerCom'2004)*, Orlando, Florida, USA, 2004.

[6] Z. Maamar, B. Benatallah, and W. Mansoor. Service Chart Diagrams - Description & Application. In *Proc. of The Alternate Tracks of The Twelfth Int. World Wide Web Conf. (WWW'2003)*, Budapest, Hungary, 2003.

[7] Z. Maamar, S. Kouadri Mostéfaoui, and H. Yahyaoui. Towards an Agent-based and Context-oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, 2005 (forthcoming).

[8] M. Nassar, B. Coulette, X. Crégut, S. Ebersold, and A. Kriouile. Towards a View Based Unified Modeling Language. In *Proc. of The 5th Int. Conf. on Enterprise Information Systems (ICEIS'2003)*, Angers, France, 2003.

[9] D. Papadis and T. Sellis. On the Qualitative Representation of Spatial Knowledge in 2D Space. *The Very Large Data Bases Journal, Springer Verlag*, 3(4), 1994.

[10] M. Roman and R. H. Campbell. A User-Centric, Resource-Aware, Context-Sensitive, Multi-Device Application Framework for Ubiquitous Computing Environments. Technical report, UIUCDCS-R-2002-2282 UILU-ENG-2002-1728, Departement of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 2002.