Event-Driven Response Architecture for Event-Based Computing

Vijay Dheap[1,2] and Dr. Paul A.S. Ward[1]

[1]Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L
http://www.ccng.uwaterloo.ca/

[2]IBM WebSphere Everyplace Mobile Portal Team
Research Triangle Park, N.C., USA

Abstract

Service-based computing is rapidly replacing the more-traditional approaches to architecting distributed systems. The critical advantage of service-based architectures is that they require only a specification of protocol, and not of API. As such, they engender a significantly looser coupling than prior techniques, thus facilitating seamless collaboration across systems and across administrative domains.

A Service-Oriented Architecture (SOA) is a middleware platform that provides a service-based computing environment. The "publish-find-bind" paradigm at the core of SOA enables the development of service-provision software separately from the development of service-consumption software. Closer observation of each aspect in this paradigm reveals that significant developer involvement is still required to assist the interaction between service provider and consumer. Developers of service-consumer software make the decision to employ a set of service providers at development time. Some SOAs provide facilities to programmatically search, bind, and even invoke services dynamically. However, it is still assumed that knowledge of both service providers and the service provided is known at development time, or the client must supply highly-detailed information about services they wish to use. This severely limits the possibility of dynamic run-time interactions among service providers and service consumers.

In this paper we introduce EDRA, the Event-Driven Response Architecture for service-based computing. EDRA is a software framework that provides an infrastructure to dynamically select client-relevant service providers during run-time. Information services selected by EDRA on behalf of clients may send notification events in case of changes in the service. In such cases, our runtime will automatically process the notification based on a selection of user-choice, system defaults, and available action services. We have implemented a prototype of our framework, and show its operation in the domain of airline services.

***Index Terms*—Event-driven systems, Web Services, Service-oriented architecture.**

Copyright © 2005 Vijay Dheap, Dr. Paul A.S. Ward, and IBM Corp. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

1 Motivation

The EDRA project was undertaken using a scenario-based development process. Concrete scenarios allow us to clearly acknowledge the problems that exist. The generalized architecture was then extracted by the specification of different scenarios, and then factoring out the common issues that all scenarios had to resolve. For the purpose of this paper, we will limit the number of scenarios we present to just a couple.

The first scenario we present is about Mary, who plans to pick up her mother from the local airport. Mary arrives at the airport 20 minutes early only to discover that her mother's flight has

been delayed by an hour. Once at the airport, Mary has very few options other than waiting. Mary may have preferred to carry out other tasks during that time, had she been aware of the flight delay.

This scenario is a realistic problem when we observe that approximately 1 in 4 flights are cancelled or delayed [11]. Studies have also indicated that increasing a traveler's awareness of delays or cancellations would improve their traveling experience [4].

Analyzing this scenario, consider how Mary could have been made aware of the flight delay. One possibility is that Mary calls the airport for information about her mother's flight before leaving for the airport. This option has three significant drawbacks:

1. The onus is placed on Mary, and if she forgets, she still suffers the wait.
2. In general flights are not late. Mary should not have to inquire to find out that "things are normal."
3. Mary will only be notified of the delay when she makes the phone call and not when the change actually takes place. This could have two negative outcomes. First, if Mary calls before any flight delay is identified, she still suffers the wait. Alternately, she may call after the flight delay is identified but sufficiently late that she cannot schedule or complete another task.

A second possibility would be for the airline company to set up a notification service. Mary can then register to be notified in the event that the flight is delayed or canceled. This option, while addressing issues 2 and 3 of the above-listed problems, still puts the onus on Mary to register for the notification. If she forgets, she is back to the original problem. If Mary does remember to register she must also remember to deregister for the notification if her mother's plans change. This can discourage users from using the notification service in the first place. Another possibility is to have a third party such as a travel agent register the notification on Mary's behalf at the time that the flight is booked for Mary's mother. Presumably the travel agent would also deregister Mary from the notification when her mother changes her plans. A third party such as a travel agent might themselves become overburdened with registering and deregistering for notifications on behalf of all their clients. It may also not be desirable for clients to allow third parties to be privy to all their actions.

1.1 Increasing Complexity

The initial scenario about Mary that involves a single notification request is relatively simple, and can probably be registered by a human in a fairly straightforward manner. However, the solutions provided cannot scale to situations that can become arbitrarily complex.

Consider the case of Simon in Connecticut for three days on business. On his last day he gets held up in a seminar and when it is over he realizes that he has to rush in order to make his flight back home. On his way to New York it starts raining and, as he approaches the airport, flights are getting canceled because of the deluge. Upon reaching the airport and giving up his rental car Simon realizes that his flight is canceled. This requires him to make his way into New York City to find a hotel for the night, since the hotels near the airport are all booked. He would either have to rent another car or use a cab. He will also have to make arrangements with the airline for an alternate flight.

Analyzing this scenario, the following observations can be made. First, it should be clear that the notification scheme has rapidly become too complex for a typical human user to take advantage of the system. Not only would Simon have to register interest in the status of his flight, but so would the car rental company, the hotel, and the airline reservation system. This would still leave open the problem of what to do in the event that Simon has already checked out of his hotel and needs a different hotel. Second, though an application built to deal with this specific scenario is not difficult, such an application would have no further use beyond this immediate problem. The invested development time cannot be justified.

The scenarios thus far introduced have been travel-based, but the problem is not limited to that domain. Any environment in which events trigger changes or transitions in a prescribed schedule can cause problems to occur to which appropriate responses must be taken.

In analyzing scenarios from other domains, described in our complete document [3], we discovered three common features:

1. A person can develop a customized solution in trivial scenarios (simple notifications, repetitive cases, *etc*.). However, most scenar-

ios are either too specific to warrant the writing of an application or evolve over time. A generic solution is called for.

2. An event-driven model applies in any environment in which events trigger changes or transitions in previously prescribed actions. Normally a certain process is followed. In the case of a divergence, various entities should be notified pro-actively. Thus, the approach can be used in health-care management, taxation systems, business-process management, conference organization, *etc*. However, for the purpose of this paper, we will focus solely on the travel scenarios, which are also the domain of our prototype implementation (see Section 5).
3. Events cross administrative boundaries. Any automated system to deal with events must likewise be able to cross such boundaries. This implies that the architecture must be loosely coupled.

The Event-Driven Response Architecture (EDRA) operates using a three-step process. It maintains the client's current and future context, and leverages that information to subscribe to relevant subscription services. These services will provide notifications when changes occur that may influence the client's current or future context. Upon receiving notifications, EDRA semi-automates a response by either following pre-defined policies or by providing a recommendation of relevant services that can be used by the client to adapt to the changes.

2 Problem Definition

Given the motivations described above, we then focused on the key constraints of any good solution. These constraints have driven the development of EDRA, and, as we describe the EDRA architecture in Sections 3 and 4, we will show how we have satisfied these constraints.

2.1 One Point of Data Entry

Information should enter the system once. The management of this information should be automated so that it flows seamlessly within the system for access and processing. Referring back to the first scenario, Mary enters information about going to the airport in her calendar and must supply it again if requesting a notification about updates on her mother's flight. This redundant effort needs to be eliminated. Its elimination is required for two reasons. First, it represents inefficiency within the system. While modest in this case, it can grow substantially in complex scenarios. Indeed, this requirement of informing the system multiple times of the same thing was one of our strongest motivations in this work. Second, and of greater significance, it can easily lead to problems and/or inconsistencies. Specifically, the requirement to enter the same information twice is also the opportunity to generate an inconsistency between different versions of what is supposed to be identical data. Rather, we propose that the information entered (once) is used to gain other relevant information without forcing the client to supply the same information multiple times.

We also note that a different type of redundancy occurs if Mary had signed up for a notification service either by herself or through a travel agent. The responsibility for deregistering is left up to the client who performed the action. Once information becomes irrelevant for any reason it should be removed from the system. Determining when a piece of information becomes irrelevant needs to be automated otherwise clients would have to enter similar information twice.

The significance of this is that it simplifies, and encourages adoption of, productivity mechanisms.

2.2 Generic Architecture

As was noted earlier, manual solutions require considerable conscious effort on the part of the client and customized solutions have severe limitations. To be responsive we are required to identify possible sources of change and all the various sources of information even though, generally, significant changes are rare. Since each scenario is unique, the development time invested to build an application to deal with changes would outweigh its benefits. Therefore, the development of an architecture that can be customized for various application domains with the flexibility to address specific scenarios is called for. The development of such a generic architecture relies on abstracting out the common functionalities. For each application domain the architecture should support "plugging-in" of customized modules. Once both the generic platform and customizable modules are assembled, the architecture should be capable of handling various scenarios within that domain.

The significance of this concept is that it makes the solution feasible and practical for adoption.

2.3 Managing Client Context

The client delegates responsibility to the system to monitor relevant changes in its environment and take appropriate responses when possible and provide notifications otherwise. To achieve this the system has to be aware of the context of the client in the present and future so that is able to find and use relevant information sources. The management aspect comes into play when a client-defined context needs to be altered because of changes that occur in the client's operating environment. A client's operating environment is defined as all the properties that constitute a context of the client, and it is the changes in these properties that have to be managed. The system makes use of the context to determine relevant service partners. The system is then capable of guiding the client through the response phase, either by automating the modification of present and/or future context based on pre-specified preferences, or through an interactive process. Referring back to Simon's scenario, the system has to be aware of Simon's present context ("in a meeting"), as well as future context ("catch a flight") in order to ascertain the information that would be relevant to him. Subsequently, when relevant information signifies changes in the operating environment of Simon's future context ("flight cancelled"), the system manages the response. The response would be either to use Simon's pre-specified instructions or to guide Simon through the response decision-making process.

The significance of this concept rests in the fact that it provides the client with a relevant structured response to reduce uncertainty.

2.4 Client Control of Context

This is an extension of the previous concept, which stresses the importance of leaving clients in control of their own context. The system will monitor only the context the client explicitly specifies, and even then does not share this context with other entities. The client makes the ultimate decision about contextual alterations due to changes in the operating environment.

The significance of this concept is that is promotes the protection of a client's privacy which otherwise might limit the adoption of the system.

2.5 Certifying Services

Currently, in service-oriented interactions, entities rely on service descriptions to specify the facilities offered by a service. However, there may be cases when the services do not offer the services they advertised. This can be caused either by genuine errors or by malicious activity on the part of service providers. This has not been much of an issue in the past, since the service provider and consumer were consciously establishing relationships. However, when we introduce the ability to automate service-relationship establishment, a mechanism to ascertain the validity of service providers is needed. An example to illustrate this point: when accessing a particular URL on the Internet which promises a certain type of content a user is redirected to content of a different type.

The significance of this concept is that it guarantees that automatic selection of service providers does not compromise reliability or dependability.

3 The EDRA Approach

Figure 1 shows a high-level overview of our architecture. The design elements within the EDRA framework can be classified into three classes: Periphery, Data Model, and Core. The EDRA Periphery is a collection of gateways through which a client can access the EDRA Core and is also responsible for mapping external data to the EDRA Data Model. The EDRA Data Model defines the type, structure, and representation of data in the EDRA Core. The EDRA Core itself can be divided into three subclasses of design elements. The first is the Data-Capture Portal, which serves as the entry point for client access to the EDRA Core. The Data-Capture Portal creates an EDRA service instance for each client. Next is the Context Container, which stores a client's data. Each Context Container is affiliated with a unique Response Platform which automates the establishment of service relationships with external entities. The Response Platform is also responsible for semi-automating appropriate adaptation responses when necessary using data stored in the Context Container. A more-detailed architecture diagram is shown in Figure 2. We now describe the subcomponents of our architecture.

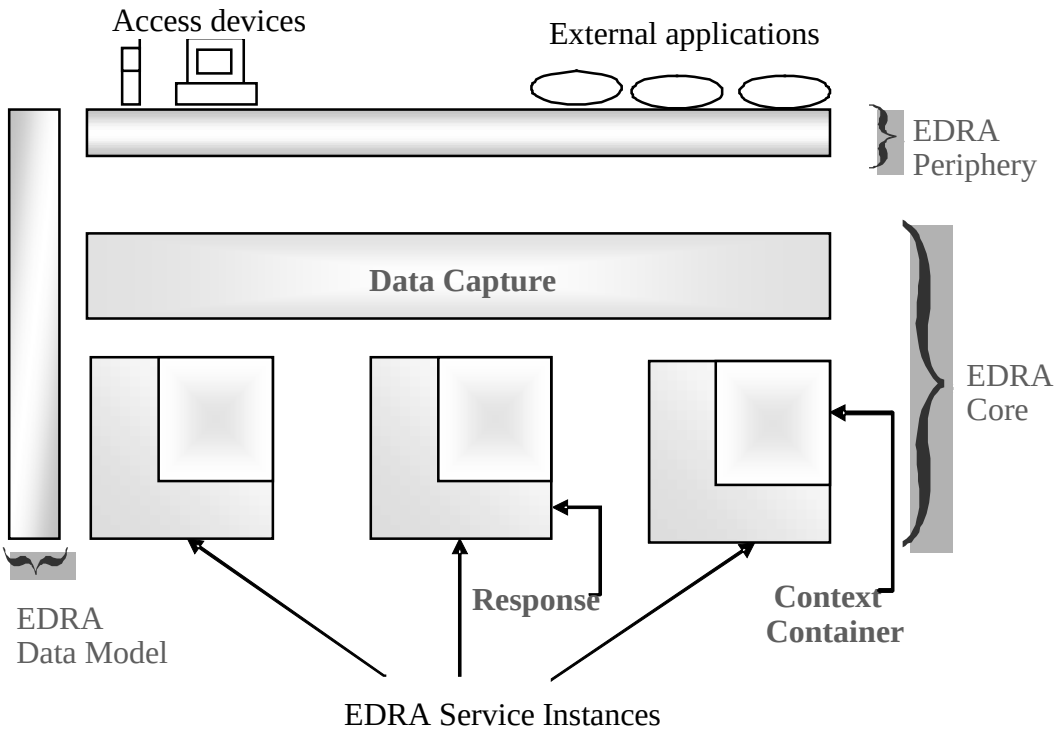

Figure 1: EDRA Architecture

3.1 Periphery

The EDRA Periphery is necessary to manage the bulk of interactions with the client by providing a single point of data entry. Client interactions that deal with creating, updating, accessing, and removing EDRA service instances are performed *via* the EDRA Periphery. These types of interactions can occur either manually by the client or through client-invoked applications. For example, a human (*e.g.*, Mary) may use his/her cell phone to access his/her EDRA service instance or invoke an application (*e.g.*, travel booking application) to update information. The EDRA Periphery handles manual interactions separately from interactions via applications. The Transcoding Manager is responsible for handling gateways that deal with manual interactions and the Compatibility Manager administers gateways geared for interactions *via* applications. The EDRA Periphery is also required to map information from clients and other external sources (*e.g.*, Applications) to the data model of the client's application domain.

3.2 Data Model

EDRA provides for loose coordination among dynamically-interacting entities by requiring standardization of data. The EDRA Data Model is tasked with maintaining the standardized type, structure, and representation of data to facilitate data processing within the EDRA framework and communication with external services. This is congruent to the standardization of data initiatives currently being seen in various domains [5,6]. The EDRA Data Model is embodied as a number of Industry Vertical Type Systems (IVTS). Every IVTS captures application-domain-specific knowledge that is not generally available outside the industry segment. Each application domain will have abstractions that generate requirements on what the data is and what it means. For example, in our travel scenario, data elements are used to describe operations (*e.g.*, flight) and generalizations (*e.g.*, vacation, business trip). The EDRA Data Model presumes that for each application domain the data can be standardized into a model. Thus, a "Flight" will be a defined type in an air-

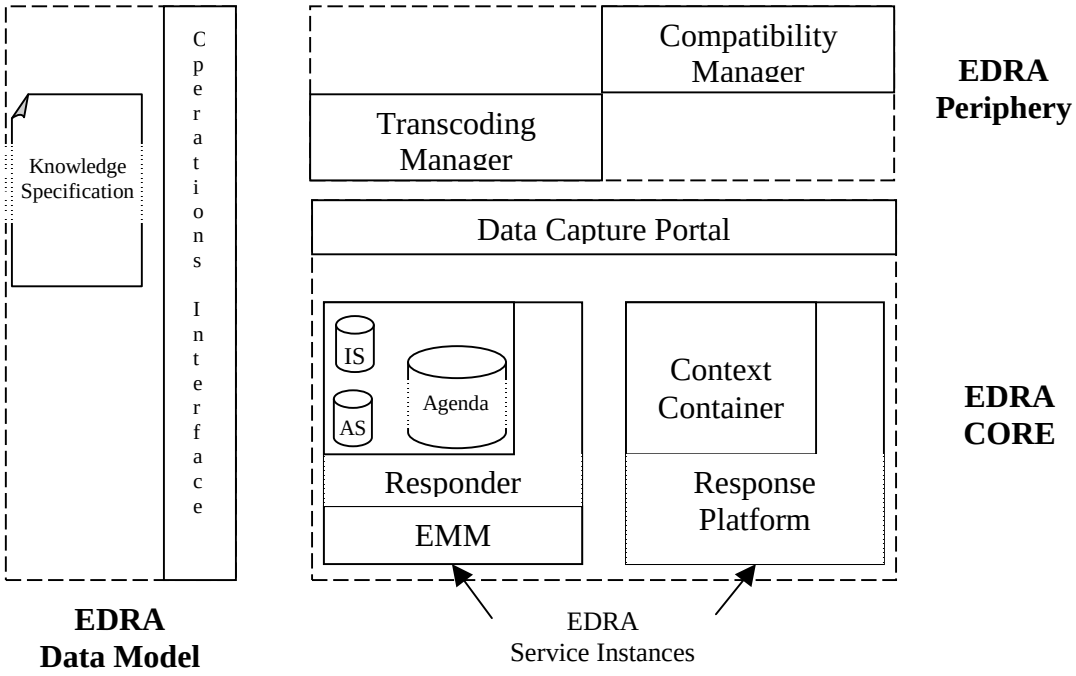

Figure. 2: Detailed Architecture

line type system, specifying such things as origin, destination, flight number, *etc*. The IVTS thus provides an interface through which other components of the framework can access data types and formats of the application domain. The EDRA Data Model also specifies the external services that are consistent with the data model associated with the client's application domain.

An IVTS is best explained by discussing its two components, the knowledge specification and the EDRA operations interface, separately.

3.2.1 Knowledge Specification

Knowledge specification has multiple purposes, but the most significant is the standardization of data within the application domain. The first construct we require in data standardization is the definition of various "simple" and "complex" types to assemble a vocabulary specific to the application domain. Next, we use this vocabulary to develop high-level abstractions that we refer to as Entries. These high-level abstractions represent the activities that a client can undertake within the target application domain [1]. The cardinality of this set of abstract activities will depend on the level of accuracy required. In the process of recognizing these abstractions we can also ascertain certain procedural relationships among them. Entries capture the properties of the context in which the client can carry out these abstract activities.

The third aspect of the knowledge specification is that it supplies a listing of all industry services that adhere to the EDRA Data Model. This is a critical feature of the knowledge specification that treats services as data pertaining to the application domain. The term service should be understood using the SOA within which the EDRA framework is deployed (*e.g.*, Web Services). Every industry service named in the listing provides a set of Entries for which the service will be relevant. This is our approach to building the infrastructure for determining relevance between a client's context and available services.

We categorize services as information or action services. Information services (IS) are the services that provide updates on changes to client's contexts (represented as Entries within EDRA). Action services (AS) allow the client to modify their current or future contexts. EDRA computes the relevant information and action services based on the client's scenario (*i.e.*, their schedule) and the IVTS for the data items in that scenario.

The final construct of the knowledge specification is to document industry-level contingency policies for each Entry. A contingency policy is a pre-specified plan for responding to anticipated changes in the operating environment(s) of an Entry. In our framework, contingency policies are a set of event-condition-action rules. EDRA also allows Entry-level contingency policies but the client specifies these, whereas the industry-level contingency policies are industry defaults, used when the clients have not defined their own.

3.2.2 Operations Interface

The Operations Interface provides other components of the EDRA framework with access to the knowledge specification.

3.3 Core

The EDRA Core is the heart of the EDRA framework. First, the EDRA Core allows new clients to register for new EDRA service instances. This allows the clients to enter current and future contextual information for monitoring and management in the form of a partially-ordered schedule. For example, in Simon's business-travel scenario his current and future context information would be his plan for the day (from the meeting drive to the airport to catch his flight using his rented car). The EDRA Core then takes responsibility for identifying and subscribing to the appropriate information sources (external services) that will provide client-relevant information. Upon receiving notifications from these information sources, the EDRA Core will either execute pre-specified responses based on the information received or notify the client. As described above, the EDRA Core can be divided into subclasses of design elements, Data Capture Portal, Context Container, and Response Platform, which together meet all the requirements placed on the EDRA Core.

3.3.1 Data Capture Portal

The Data Capture Portal simply provides a global access point to the EDRA Core. It handles the initiation of every EDRA service instance for a client and does this multiple times in a multi-client deployment environment. Creating new service instances involves associating a new Context Container instance with a new Response Platform instance.

3.3.2 Context Container

All the information that a client enters into his/her EDRA service instance is stored in the Context Container. The types of information about a client that is stored in the Context Container include registration information, current and future context, a list of services that are relevant to the client, and pre-defined policies for responding to changes.

The most critical subcomponent is the Agenda. The Agenda maintains the client's current and future context. Recall that the format of the data stored about the client's current and future context is governed by the EDRA Data Model. The Agenda therefore contains a set of Entries within that model.

Dependencies among these Entries are captured by the Agenda's Dependency List, forming a partial order. The agenda can then be interpreted as a schedule for the client. This is critical, because to preemptively employ services we need to know the current and future operating environments depicted as Entries to determine relevance. Furthermore, the set of Entries in the Agenda and their order allow the client to define any specific scenario within the client's application domain.

The client can specify pre-defined contingency policies at the Agenda level and at the Entries level. For each entry a client may specify what the appropriate response should be for a given notification received from a relevant information service. The client also has the flexibility to define appropriate responses at the Agenda level, possibly importing corporate contingency policies.

The Context Container has two Service Repository subcomponents. One holds all information services that have been subscribed to based on the Entries in the Agenda. The second holds all the action services that can be used to modify the Entries in the Agenda. Each service stored in either repository holds references to the set of Entries for which it is being used.

3.3.3 Response Platform

The IVTS provides the infrastructure to ascertain the relevance between Entries and services. The Response Platform now adds the infrastructure to subscribe to relevant information services. The Response Platform is also the infrastructure required to semi-automate client responses to notifications received from subscribed information services by identifying the relevant action services.

The Responder subcomponent manages the activities of the Response Platform. It is responsible for managing service relationships with relevant information services and, when necessary, action services. It also manages the execution of contingency policies based on the event notifications it receives from the Event Management Module.

The Event Management Module (EMM) subcomponent is responsible for establishing dynamic transient service relationships with external information services. The EMM manages the subscriptions and funnels the event notifications back to the Responder.

4 Behavioral Design

We now trace through our travel scenario and describe how the EDRA architecture determines what services a client should subscribe to, and what actions it must take in managing its current and future contexts. The purpose of this section is thus to identify and discuss the set of behaviors that govern how the structural components discussed in the previous section operate and interact to provide a cohesive solution.

4.1 Simulation of the Scenario

Before discussing the behavioral aspects of EDRA we must first outline our simulation environment that was used to develop these behaviors. The first step was the creation of a sample IVTS for the travel industry. For our purposes, we implemented the EDRA Core as a hosted web service that enables the client to create and manage their EDRA service instances. The EDRA Periphery consists only of a Transcoding Manager that presents an HTML-formatted user interface for the EDRA Core web service. External information and action services were also implemented as web services. It should be noted that these services are referenced by the IVTS we created as required by the EDRA Data Model definition.

To simulate a scenario a discrete-event simulation model was used. An external time web service notified the EDRA Core web service and the information services involved. This event triggered a sequence of actions on the part of an information service or the EDRA Core web service. For example, the flight status information service was programmed to generate a flight delay status update at a specific time event generated by the time service.

With this simulation framework we can describe the behavioral design of the EDRA solution using Simon's travel scenario as an example.

4.2 Access

Clients first require the ability to access the EDRA Core. Clients initiate the interaction when they want to register for a new EDRA service instance, or to add, update or remove Entries from the Context Container associated with an existing EDRA service instance. We also stated that client access could occur either through devices or applications. Providing clients with the flexibility to interface with EDRA using their preferred mode is fundamental to achieving one point of data entry. It allows them to adopt the EDRA solution without having to change their previous processes. The Data Capture Portal of the EDRA Core listens for new connections. It manages a new connection initiated by an access device through a Transcoding Manager. A new connection can also be initiated by a Compatibility Manager acting on behalf of an application.

In our scenario, Simon accesses an EDRA service to register. Likewise, his travel application accesses the service to populate his EDRA service instance with his travel plans. Finally, while on the trip, Simon might need to access it to update or verify aspects of his itinerary.

4.3 Registration

A client must be known to EDRA before it can assign a service instance to that client. Registration behavior is responsible for governing this type of interaction. The first access made by a client to EDRA would be to initiate the registration behavior. Registration provides the process for identifying the client so that future access for that client's EDRA service instance can be authenticated.

During registration EDRA will request information about the client and this could vary for each application domain. The information requested by EDRA is defined at deployment time of the EDRA solution. The minimal amount of information required by EDRA is the client's name, possibly a unique identifier, and contact information. In our business travel scenario, Simon could be requested for his air miles card number or his car rental loyalty card number. This information will be used by the EDRA service instance to interact with external services on behalf of the client.

The registration process allows the client to identify the external applications from which they will transfer information to their EDRA service instance in the future. An appropriate Compatibility Manager must be set up with these applications. As an example, Simon would identify the travel application that constructs his travel itinerary so that it can automatically transfer data to his service instance.

4.4 Acquisition of Contexts

After registration, a client has an EDRA service instance that can monitor and manage their current and future contexts. It is necessary for the client (of the EDRA service instance) to provide EDRA a preliminary list of contexts in which the client is expected to be over a period of time (internally represented as Entries). In our scenario, this is Simon's travel itinerary. Only a client, or the owner acting on behalf of the client, has this knowledge. An EDRA service instance can only be preemptive in assisting the client upon knowing the intentions, or plans, of the client in the form of their expected contexts. When the operating environments that constitute the client's contexts change, EDRA can exhibit preemptive behavior.

An EDRA service instance can acquire the preliminary set of contexts for the client either from specialized applications (*i.e.*, Simon's travel application) or manual entry by the client/owner using an access device. When entering new contexts manually, the Transcoding Manager uses a form-based approach to guarantee that the data fed into an EDRA service instance corresponds to the IVTS of the application domain. Updates can also be made to modify or remove a current or future context in an EDRA service instance.

4.5 Initialization

This behavior defines how an EDRA service instance prepares itself for monitoring and managing behavior after acquiring the client's current and future contexts. The main preparation that has to be made is in the automatic selection of relevant services based on the client's current and future contexts.

Since each context has an equivalence relationship with an Entry in the IVTS for the application domain, the knowledge specification of the IVTS can be searched for services that are deemed relevant to that Entry. Specifically, we can compute, based on the type of the Entry, and its instantiated values, what services are required. For example, if Simon has booked a flight, the type information of "flight" will include a notification services for changes on that flight. The specific service to subscribe to will be based on the specific flight, which is in the instantiated data. Thus, if the flight is American Airlines 892 from New York to Boston, on June 19th, the American Airlines notification service will be subscribed to, with the relevant parameters. Likewise, Boston weather information will be subscribed to.

These relevant services will be maintained along with the Entries representing the current and future contexts of the client. Referring back to Simon's business trip, once his itinerary is entered into his EDRA service instance, the Travel IVTS is searched to find all the information and action services that will be relevant to each of his contexts. Examples of these services could include services related to flight information, weather, hotel reservation, flight reservation, *etc*.

4.6 Monitoring and Management

This behavior is realized through three sub-behaviors. The first is the Information-Service Relationship Establishment, which is responsible for establishing dynamic service relationships with relevant information services. This is the means by which this EDRA instance subscribes to the relevant information services selected during initialization. The second behaviour is Channeling Events, which directs updates received from information services to appropriate contexts being Monitored and Managed by the EDRA service instance. The third is Transition, which defines how an EDRA service instance internally models

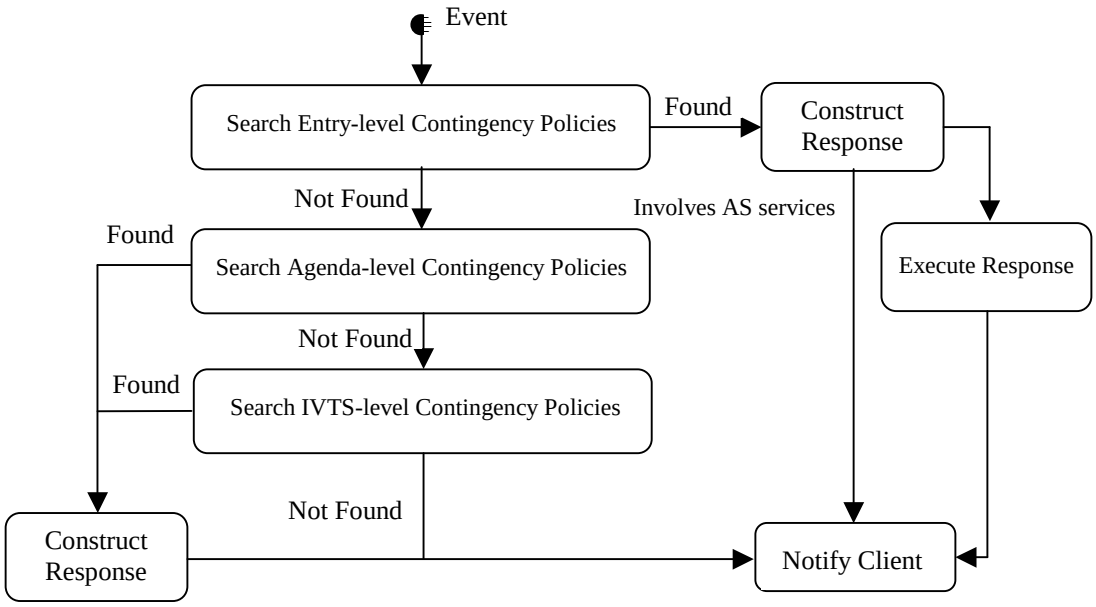

Figure 3: Response and Adaptation

the transition from one context to another. For simplification of the Transition sub-behavior we have assumed the existence of start and end times for each context. Together these sub-behaviors provide the means to preemptively identify changes in a client's schedule.

As Simon proceeds through his business trip various things may change: the weather conditions, flight delays, availability of rental cars, and possibly even the duration of his meeting. The relevant information services that are subscribed to by the EDRA service instance managing and monitoring Simon's business trip allows it to build awareness. Events are published by these information services so that an action can be taken if necessary. It is also entirely possible that nothing of significance changes in Simon's trip in which case the EDRA service instance simply needs to mimic Simon's transition from one context to another (*e.g.*, leaving the meeting and going to the airport).

The final behavior, Response and Adaptation, defines how an EDRA service instance can preemptively automate or semi-automate a reaction to changes in contexts of the client.

4.7 Response and Adaptation

Figure 3 illustrates the decision-tree used for response and adaptation behavior. We make use of the responses stored in the contingency policies for anticipated changes in the environment.

If, for example, Simon's flight was cancelled, and he maintained a contingency policy to reschedule the booking, then the following sequence of actions would occur:

1. The flight information service would report the cancellation to Simon's EDRA service instance.
2. The Response Platform of the EDRA service instance would attempt to find a contingency policy (Entry level, Agenda level, and finally IVTS level)
3. Execute the contingency policy, which in this case involves a flight-reservation service.
4. Since the flight-reservation system is an action service, Simon is notified of the situation and provided with the option of different flights (from the flight-reservation service) that may suit his need.
5. Simon makes the ultimate decision to proceed with the reservation.

5 Prototype Implementation

As we have already noted in Section 4.1, we have implemented a prototype of the EDRA core, as well as a highly-simplified IVTS specification of the travel industry so as to validate our architecture. The EDRA Core was written as a J2EE application running under the IBM WebSphere Application Server. Its operation is essentially as described in Sections 3 and 4. The simplified travel IVTS knowledge specification was created using XML Schema-Definition Language. It provides an XML hierarchy for basic transportation types (air, train, bus), together with various hotel types. Within these categories are appropriate subdivisions (*e.g.*, airline carrier, locations, *etc*.). Contingency policies, as well as information and action services were similarly defined.

To drive this system we wrote a small browser-based client, which acts as the EDRA Periphery. This client allows registration of the trip, in *lieu* of actual integration with existing client applications. After the system is initiated, change from the prescribed schedule is made possible by having the information services alter the normal course of events.

We note that our prototype is substantially limited. In particular, the Core is far from scalable, operates the agenda as a simple dependency list, and will only operate with a single IVTS. Integration with existing applications is lacking, and the IVTS is far from comprehensive. However, within the limitations of our prototype, we found it operated as required by our design, and serves as an adequate proof-of-concept. In particular, it demonstrates that automated selection of relevant services is possible, and semi-automated responses to generic scenarios are feasible.

Further details of our prototype are available in the complete document [3].

6 Related Work

Semantic Calendars are an AI approach to solving the specific problem of scheduling meetings among a number of entities. Restina Semantic Calendar Agent [9,10], or RCal, is one such project that aims to automate the selection of a mutually-convenient time for a meeting. Management of the user's calendar entries and automating responses to changes is not within the problem domain for RCal. RCal also has not focused on specifying or taking into account the user's context in the process of scheduling meetings. Contextual information could play an important role in the successful scheduling of meetings.

Paar and Tichy [8] outline their approach to incorporate semantic processing into Web Services infrastructure. Their aim is to enable the selection of service providers and to execute operations of their selected services during runtime. One aspect of the problem it does not address is that when automatically selecting services, a level of trust has to be developed between the service provider and service consumer. When services providers were selected during development time there was a conscious decision made about trusting the provider. In interactive situations, where client involvement in specifying search criteria (*e.g.*, natural language description) and selecting services is required, the advantages of this approach would be mitigated due to its lack of proactive behavior. This level of support, albeit with a lower degree of accuracy, can be provided without semantic annotations using syntactic matches, à la Google.

Context Service [7] is a service-based approach for integrating context-awareness into applications. The context service is a robust and modular approach to designing context-aware applications. Its aim is to find and manage the contextual information about subjects (people or objects) about which its client is concerned. The context service only retrieves context information when queried by the application. Changes in the contextual information are not proactively propagated, which is required for event-driven applications. In addition, it does not take into account future contexts of the client, which may be required by applications that need to respond to contextual changes (*e.g.*, if the flight is cancelled then Joe cannot make the meeting). Context service is a middleman between context sources and applications that need context information. Incorporating new sources can only be done at development time, which restricts the dynamic selection of information sources during run time. The process of leveraging the information delivered by the context service is beyond the scope of this service and is left up to the application invoking it.

The iQueue Project [2] was geared to address the issues surrounding data composition in reactive applications. It focused on optimizing aggregation of information from various sources, concentrating mainly on the quality of service

offered by these sources for selection to meet functional data specifications. In iQueue it is assumed that the types of information required are known, and a composer is generated to manage the selected information types. The framework does not outline the process in which the information types themselves were selected. At present, iQueue also does not identify the manner in which the initial set of member-data sources a composer uses is selected, and is the topic of future research. Since application developers define the behavior of composers at development time, they must also predetermine the types of information they require, reducing the applicability of this framework in dynamic applications where relevant data sources may change over time.

7 Conclusions

A set of basic objectives was established for the Event-Driven Response Architecture that enabled us to address the five research issues. Specifically, they were:

1. a mechanism for automating the selection and invocation of client-relevant services during runtime;
2. non-intrusive and simplified information gathering method;
3. a process to enhance the reliability and dependability of automatically selected services;
4. a preemptive approach to identifying changes that may influence the client's current or future actions; and
5. a mechanism that enables clients to use a policy-based procedure to semi-automate responses to changes that may influence their current or future actions.

The contributions of the Event-Driven Response Architecture are a direct result of our approach to achieving the objectives stated above. In particular, our four major contributions are:

1. proposing a minimum level of standardization required in service-based computing;
2. outlining the infrastructure required to achieve an understanding of relevance in terms of the client;
3. defining the infrastructure for preemptive identification and response construction for changes that may influence the client's current or future actions; and
4. ensuring that the framework is customizable, with the necessary flexibility to work across various industries and application domains.

We have validated our contributions by implementing a prototype of our framework. This prototype has demonstrated that relevant services can be computed when data types are standardized, and responses to changes can be automated or semi-automated, again based on industry-specific standardization.

Moving forward there are two areas of the framework were the simplifications can be removed. First, given the significant role played by the EDRA Data Model within the EDRA framework, we want to explore the feasibility of creating an XML-based extensible language for defining the knowledge specification. Standardization in defining the knowledge specification would give us the ability to develop tools to rapidly model data in an application domain. While considering this we may also want to delve into how a single EDRA solution can be made to operate using multiple IVTSs. This would enable clients to use the same EDRA solution for separate application domains they may operate in. For example, a person may require EDRA to monitor their contexts beyond just their work environment. During the day, while at work, EDRA could also monitor their home environment. If two different IVTSs are defined one for the work environment and one for the home environment then the EDRA solution must be able to process Entries from both. In particular, it must be able to resolve conflicts that occur between the two environments.

The second challenge to be tackled is the representation of a client's current and future contexts in the Agenda subcomponent of the Context Container associated with an EDRA service instance. This is one of the improvements we have alluded to before. Entries in the Agenda should not have to be just time-based, and we need a more robust mechanism than the dependency list to describe ordering relationships among Entries. Tracking client transitions from one Entry to another would also fall within the realm of this initiative. This would prove very important in application domains such as business processes or workflow monitoring.

Acknowledgements

The authors would like to thank Dr. Arthur Ryman of the IBM Toronto Lab. for many productive discussions regarding this work.

About the Authors

Vijay Dheap is currently a member of the IBM WebSphere Everyplace Mobile Portal Team based out of Research Triangle Park, N.C., USA. He graduated with a Masters in Computer Engineering from University of Waterloo, Canada and was a student researcher in the field of pervasive computing at IBM CAS Toronto.

Dr. Paul A.S. Ward is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Waterloo. His research interests lie at the intersection of distributed systems and networks. In distributed computing his work focuses on distributed application management, and more generally in dependable distributed systems. In networks his interest lies in wireless data networks, and more particularly in *ad hoc*, wireless mesh, and delay-tolerant networks. The combination of these interests has led him to study problems in mobile, wireless, and pervasive computing. Prior to pursuing his Ph.D. he worked in both the hardware and software industries, covering the range from electronic parking meter design to developing the fast parallel load utility for the DB2 database system. He is a member of the IEEE, including the Computer and Communications Societies, as well as a Professional Engineer.

References

[1] Larry Arnstein, Chia-Yang Hung, Robert Franza, Qing Hong Zhou, Gaetano Borriello, Sunny Consolvo, Jing Su. Labscape: A smart environment for the cell biology laboratory. *IEEE Pervasive Computing Magazine*, vol. 1, no. 3 (July-September 2002), pages 13-21. IEEE Computer Society, 2002.

[2] Norman Cohen, Apratim Purakayastha, Luke Wong, and Danny L. Yeh. iQueue: A pervasive data composition framework. *Proceedings of the Third International Conference on Mobile Data Management*, pages 146-153. IEEE Computer Society, 2002.

[3] Vijay Dheap. *EDRA: Event-Driven Response Architecture for Service-Based Computing.* M.A.Sc. Thesis, University of Waterloo, Waterloo, Ontario, Canada, 2004.

[4] Phillip Harper. Informed Travelers = Fewer Flight Delays. Microsoft bCentral. http://www.bcentral.com/articles/harper/129.asp, accessed: May 2004.

[5] Health Level Seven, Inc. Health Level Seven. http://www.hl7.org/, accessed: May 2004.

[6] Kentucky Department of Education. Data Standardization. http://www.education.ky.gov/KDE/Administrative+Resources/Data+and+Research/Data+Standardization/default.htm, June 2005.

[7] Hui Lei, Daby M. Sow, John S. Davis, II, Guruduth Banavar, and Maria R. Ebling. The design and applications of a context service. In *Mobile Computing and Communications Review*, vol. 6, no. 4. (October 2002), pages 45-55. ACM Press, 2002.

[8] Alexander Paar and Walter F. Tichy. Semantic software engineering approaches for automatic service lookup and integration. In *Autonomic Computing Workshop: Active Middleware Services* (June 2003), pages 103-111. IEEE Computer Society, 2003.

[9] Terry R. Payne, Rahul Singh, and Katia Sycara. Calendar agents on the semantic web. *IEEE Intelligent Systems*, vol. 17, no. 3, pages 84-86, May/June 2002.

[10] Terry R. Payne, Rahul Singh, and Katia Sycara. RCal: A Case Study on Semantic Web Agents. In *The First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, ACM Press, 2002.

[11] US House of Representatives. *Flight Delays and Cancellations Continue as Major Sources of Customer Dissatisfaction*. Washington, 2000, Accessed: May 2004. http://www.house.gov/transportation/aviation/issues/delays.pdf.