EDRA:
Event-Driven Response Architecture for Service-Based Computing

by

Vijay Dheap

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Computer Engineering

Waterloo, Ontario, Canada, 2004

Borrower's Page

Acknowledgements

I want to thank all those who made the completion of this thesis possible. I want to thank my supervisor, Dr. Paul Ward, for valuable suggestions and corrections. I thank Dr. Arthur Ryman for fast-tracking my initial research efforts.

I am indebted to my family for their love, constant encouragement and judicious advice. Last but not least, I want to thank my sister, Sudha Meghan, for her persistent pestering.

Abstract

Service-based computing is fast emerging as popular area for research and commercial initiatives. Loose-coupling among entities when engaging in service-based interactions is seen as the facilitator for achieving 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 is at the core of SOA. This paradigm enables the development of service provision software separately from the development of service-consumer 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 service providers 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.

This thesis introduces EDRA: 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. 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 used by the client to adapt to the changes.

The EDRA framework is SOA agnostic and has been designed with the flexibility to be customized for various application domains (e.g. Travel-plan management, business-process monitoring and execution, patient-care management in healthcare institutions, etc.). During the development of EDRA various deployment strategies including incremental adoption were considered. The EDRA framework can potentially be applied in constructing pervasive computing solutions.

Table of Contents

List of Figures

Chapter 1

Introduction

The Internet has been rapidly transformed from a rudimentary collection of static web pages to a more dynamic, service-oriented marketplace. The pervasiveness of the Internet enables entities to enhance communication and collaboration through ease of access and a high degree of standardization. Simultaneously, the emphasis on distributed computing is steadily gaining momentum as a cheaper and/or more practical alternative for service provision and consumption. Distributed computing allows the reuse of software modules in constructing specialized applications, harnesses computing and developer resources among different groups and potentially can provide a level of robustness that centralized computing cannot. As we have been observing, the logical result of these two trends is the use of the Internet and its associated technologies as a distributed-computing platform. This thesis aims to address some of the issues that continue to arise from the merger of these two major trends. At present, Web Services technology is being promoted as the frontrunner for web-based distributed-computing enablement. Web Services leverages service-based computing concepts to provide for loosely-coupled interactions among coordinating entities.

Distributed computing and Service-Oriented Technology will be discussed in greater detail in Chapter 2. In this chapter, we will first motivate the need for an Event-Driven Response Architecture (EDRA) for service-oriented distributed computing. EDRA was designed to provide additional interaction models among entities, flexibility to address multiple application domains and various scenarios within those domains with a higher degree of automation. This chapter will also outline our contributions and the approach taken to achieve them.

1.1 Motivation

The purpose of this section is to provide a high-level understanding of the issues that EDRA is geared to address. The underlying concepts of EDRA will also be highlighted. A scenario-based approach will be used to underscore the motivations for developing EDRA.

1.1.1 Motivating Scenarios

The first scenario 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 [30]. At the time of writing this thesis several news bulletins focused on this problem:

"US Airways warns of East Coast flight delays… Charlotte Observer, NC - 26 Jan 2004"

"*Terrorist probe keys on routes, More flight cancellations, delays ... Hampshire Gazette, MA - 3 Jan 2004*".

Studies have indicated that increasing a traveler's awareness of delays or cancellations would improve their traveling experience [11].

Returning to analyze the example 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 – Mary calls before any flight delay is identified so she still suffers the wait or she calls 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 of all their actions.

1.1.1.1 Increasing the 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. Figure 1.1 shows the Simon's initial plan. 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 have to either rent another car or use a cab. He will also have to make arrangements with the airline for an alternate flight.

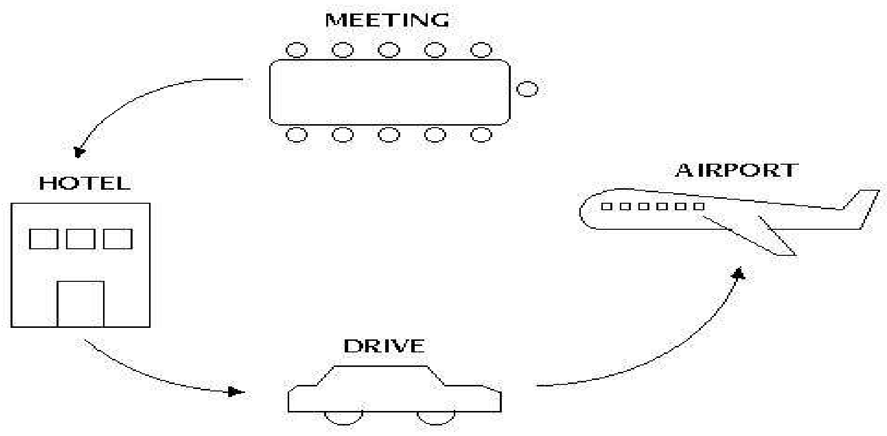


Figure 1.1 A travel scenario with multiple notifications

In this scenario the required interactions are considerably more complex. If Simon had been notified as the information became available, he might never have returned the rental car, and possibly could have extended his hotel booking in Connecticut. If he had already left Connecticut by the time he was notified of the flight cancellation, he would have instead required a hotel booking in the nearest available hotel. Thus, not only must Simon be notified of the problem, but so must the car rental company. Either the hotel he was staying in must be notified, or a new hotel room must be procured, and that decision will be context dependent. Finally, a new flight reservation must be made on Simon's behalf.

To analyze 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. Third, 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.

1.1.1.2 Another Application Domain

An example scenario in the business-process application domain will now be presented to demonstrate the generic nature of the problem. Business processes are inherently event-driven.

Acme Widget Company has four stores, two warehouses, and five products to sell. One possible business process within Acme Widget Co involves the ordering of the appropriate products when inventories drop in the warehouses and to ship products from the warehouses to the stores when shelf space becomes available. The elements of the business-process schedule are: Order (from suppliers), Stock (in warehouses), Ship (to stores), and Store (items in stores).

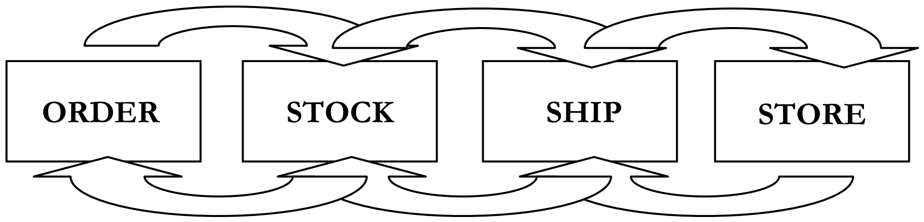


Figure 1.2 A simple representation of a Supply Chain Business Process

This business process, like others, is cyclical. Thus when the process is in Store and it recognizes that the supply of products in the stores is depleted, it reverts to Ship more products from the warehouses, and so forth. To keep track of status of this business process a customized system could be developed. One reason why it is actually feasible to construct a system to manage a business process is because they are relatively more long-lived than a travel schedule. However, services associated with a business process may change or be upgraded quite frequently to enhance operations. Over time the business process itself may require the incorporation of additional warehouses, more suppliers for additional products, and even organizational reporting changes within Acme Widget Co. This would require development work for reconfiguring the monitoring/management system each time. Even if additional data acquisition services become available the system needs to be manually reconfigured.

Here is a summary of the justifications for constructing EDRA from the motivating scenarios:

1. A client can develop a customized solution in trivial (simple notifications and/or repetitive) scenarios. However, most scenarios 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.*
3. Events cross organizational 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.

1.2 Event-Driven Response Architecture (EDRA)

We propose the Event-Driven Response Architecture (EDRA) as a self-contained platform for semi-automating service-based processing in loosely-coupled distributed computing systems. Loose-coupling requires both synchronous and asynchronous communication mechanisms for scalability. Our architecture can also be leveraged to support dynamic transient service relationships among entities. The dynamic nature of the architecture refers to the establishment of service interactions among the entities, while the transient nature refers to the temporal applicability of the service relationships.

Given the motivations stated in the previous section, it is important to present the research issues, which we refer to as generalized concepts, which have driven the development of EDRA. These concepts are:

- One Point of Data Entry – client need not have to enter the same data multiple times into the system
- Robust Architecture Customized For Various Applications – extensible system architecture that identifies commonalities across application domains
- Managing Present and Future Client Context – determination of relevant information and semi-automating responses on behalf of the client
- Client Control of Client Context – context of the client is retrieved from the client
- Certifying Services Employed – automatic selection of services requires service providers to be certified for the services they claim to provide

The above concepts are broad aims and in this thesis EDRA will be designed as an extensible architecture capable of evolving to fulfill the various aspects of these concepts. The concepts are further described below.

1.2.1 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. The information has to be used to gain other relevant information without forcing the client to supply the same information multiple times. 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 of 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 concept is that it simplifies, and encourages adoption of, productivity mechanisms.

1.2.2 Robust Architecture Customized for Various Applications

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 out-weigh 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 an 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.

1.2.3 Managing Present and Future 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 structured response to reduce uncertainty.

1.2.4 Client Control of Client 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.

1.2.5 Certifying Services Employed

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 that are advertised in its description. 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 is 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.

1.2.6 Objectives and Contributions

The basic objectives of Event-Driven Response Architecture enable us to address the five concepts outlined in the previous subsection. Specifically, it provides:

- a mechanism for automating the selection and invocation of client-relevant services during runtime;
- non-intrusive and simplified information gathering method;
- a process to enhance the reliability and dependability of automatically selected services;
- a preemptive approach to identifying changes that may influence the client's current or future actions; and
- 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.

- EDRA proposes a minimum level of standardization required in service-based computing.

- EDRA outlines the infrastructure required to achieve an understanding of relevance in terms of the client.
- EDRA builds the infrastructure for preemptive identification and response construction for changes that may influence the client's current or future actions.
- EDRA is a customizable framework that has the flexibility to work across various industries and application domains.

1.3 Approach

EDRA was initiated using the scenario-based development process. The major requirements and concepts for our architecture were marshaled from the issues that were identified in travel scenarios. In the process of generalizing the applicability of EDRA, various other scenarios were considered including business-process specification, patient management in healthcare institutions, and student information systems (e.g. UW Quest). We refined the design from the requirements of these additional scenarios. The challenge was in abstracting out the commonalties among these various scenarios. Finally, using those abstractions, we specified the algorithmic methodologies through which the structural elements of EDRA interact with each other to provide an integrated solution. A significant portion of the research involved in constructing EDRA could be described as an exercise in software engineering. As a result, we made extensive use of software design patterns to assist in the design, and documentation of the framework. We emphasized modularity, flexibility, and maintainability in every feature of EDRA.

Figure 1.3 shows a high-level overview of the EDRA framework. All the design elements within the EDRA framework can be classified into three classes: EDRA Periphery, EDRA Data Model, and EDRA 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 also responsible for semi-automating appropriate adaptation responses when necessary using data stored in the Storage Container.

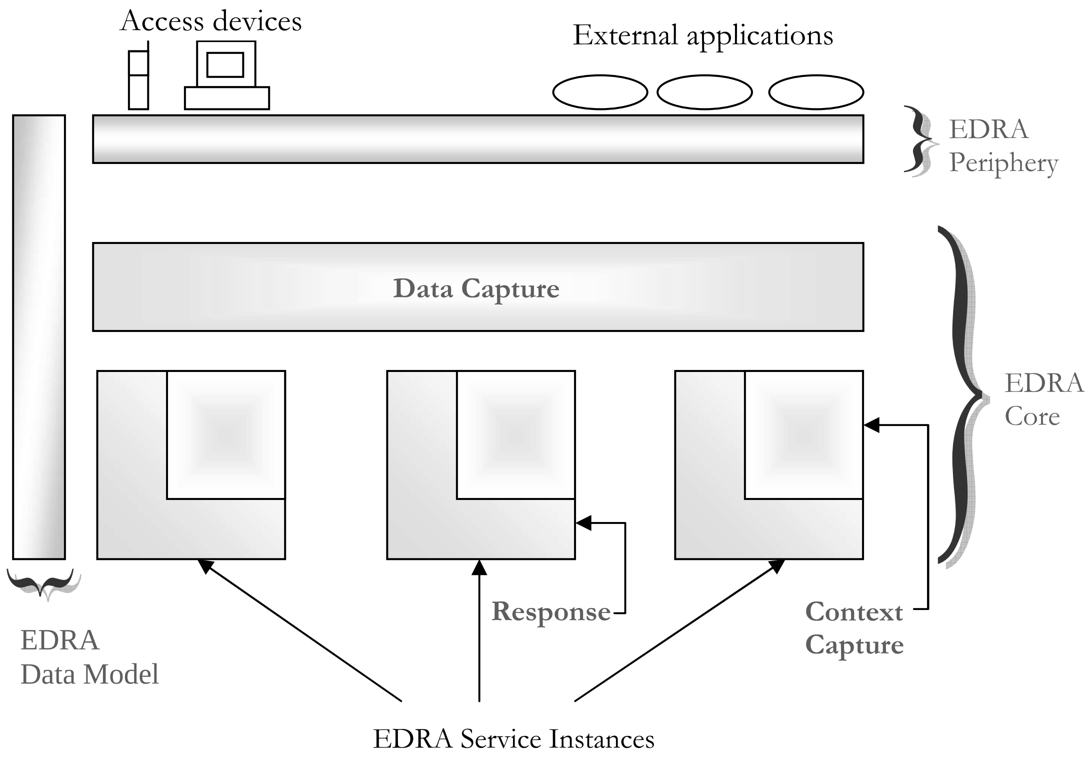


Figure 1.3 A high-level overview of EDRA

A final comment about the approach to constructing EDRA is that emphasis was placed on taking into account its adoption as a solution. We expect adoption of this solution to be incremental and will grow as the number of available services abound. EDRA was geared to building a distributed computing platform requiring minimal dependencies among interacting entities. As a consequence, EDRA assumes only one point of standardization, and that is in data. The design of the framework focused on enabling every inclined client to employ EDRA for their application domain without the need for other parties to adopt the solution or provide auxiliary support. The framework is meant to be deployable using current technologies available in service-based distributed computing (e.g. Web Services). However, EDRA is constructed such that it can incorporate future advances in these technologies (e.g. Semantic Web Services).

1.4 Outline of the Thesis

This chapter served as an introduction to EDRA, from the motivation to its contributions. In Chapter 2 the background and related work will be presented, with an emphasis on distinguishing EDRA from prior works. Chapter 3 will fully describe EDRA from a structural design perspective, discussing the elements of the framework and justifying the design choices made. The purpose of Chapter 4 is to provide a behavioral design perspective of EDRA and substantiate the framework by tracing through a case study using the EDRA framework. Chapter 5 will concentrate on reviewing the design and how it meets the objectives for the EDRA framework. The review also draws attention to how the design impacts deployment considerations. Finally, Chapter 6 will conclude this thesis by summarizing the accomplishments of the EDRA project and provide direction for future work on the project.

Chapter 2

Background and Related Work

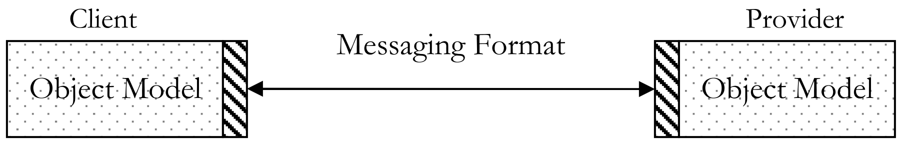
The aim of this chapter is to provide the reader with an understanding of the research context of our work, and then further clarify and differentiate the goals and aspects of EDRA from prior efforts. First, we will provide an overview of the research areas, and then move on to previous projects and published literature that are pertinent to our work.

2.1 Distributed Computing and Service Oriented Architectures

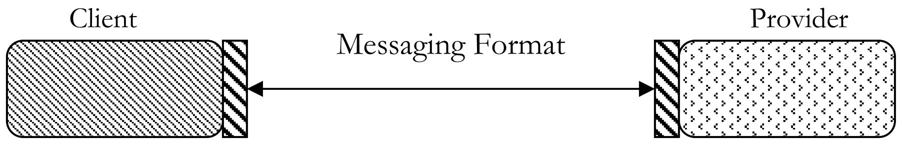
The aim of distributed computing systems has traditionally been to harness the computational capacity of multiple, independent computers, but provide the user of the system with the illusion of interacting with just one [28]. The maturing of networking technologies has made distributed computing a viable alternative to centralized systems. One of the main challenges still being addressed in distributed systems is supporting interoperability among various computers across administrative domains while still providing a unified system view.

Service-Oriented Technology (SOT) involves a set of services that communicate with each other to complete a task. Each service provides a function that is clearly defined, self-contained and its operation is independent of other services. SOT is an refinement of Object-Oriented Technology (OOT). OOT has focused on defining entities and specifying their construction, though at an abstraction level it denotes the operations an entity provides. SOT recognizes this and explicitly limits itself to focus on the specification of the operations an entity can provide [3]. Both OOT and SOT require a standard communication format for exchanging data. Pure service-oriented architectures (SOA) utilize SOT exclusively and set no preconditions on the existence of a common object model for service interactions [2]. Each client in a service relationship only has to be aware of operations available and the structure of the messages exchanged.

A distributed computing solution must transparently provide for multiple interaction models (e.g. synchronous and asynchronous), fault tolerance, transaction support, security, concurrency, and persistence of data [28]. To address these challenges of distributed computing, several OOT-based solutions have been proposed. CORBA and DCOM are attempts to address the issues of interoperability among separate systems by providing a common distributed-computing-middleware platform. CORBA and DCOM also incorporate SOT within their architecture to define interactions among objects. Jini and Salutation are SOT coordination frameworks for networked devices. More recently, Web Services based on the XML-data-exchange format are being proposed as a universal SOA platform [2]. Brief descriptions of these various systems are given as a background in the evolution of service-based distributed computing.



a. Object Oriented Technology
Two points of standardization



b. Service Oriented Technology
One point of standardization

Figure 2.1 OOT vs. SOT

2.1.1 CORBA

The Common Object Request Broker Architecture (CORBA) is an open specification of an object-based distributed system [28, 21]. It provides language and platform independence for interactions among distributed objects. Interaction among CORBA objects is handled by an Object Request Broker (ORB). The initial design of CORBA was aimed at specifying the API for the ORB, which led to a symmetrical requirement. This symmetrical requirement forces communicating entities to have the same implementation of the ORB. Initially different vendors used proprietary communication protocols, negating interoperability gains made by a standardized API. This problem was addressed through the introduction of the Internet Inter-ORB Protocol (IIOP) [20]. CORBA employs SOT to specify facility services that can either be specific to certain applications or generic. CORBA objects use these various services along with the ORB for carrying out distributed tasks. CORBA application objects are implementations of a set of CORBA interfaces. In terms of SOT those interfaces outline the services an object offers. The interfaces are defined in the Interface Definition Language which maps the specification to a programming language.

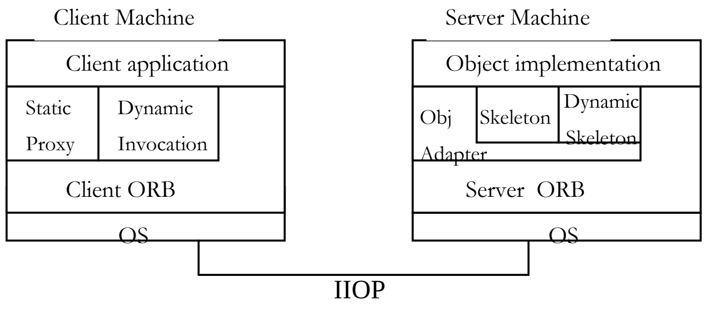


Figure 2.2 CORBA distributed computing specification

2.1.2 DCOM

Distributed Component Object Model (DCOM) is Microsoft's extension to provide distributed-computing support for its Component Object Model (COM) [28, 17]. COM objects are components that encapsulate a specific functionality. Each COM object has one or more interfaces through which its functionality can be accessed. COM objects are designed to interact with each other to provide dynamic support for message passing among various applications each with possibly different types of data. DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP so that remote access to COM objects on different computers becomes possible. Services in DCOM include the operations that COM objects support and standard facilities provided for clients. One of DCOM's limitations is that it is highly tied to the Microsoft Windows platform and the services the operating system provides applications (e.g. Active Directory).

It is important to note that in CORBA and DCOM both the service provider and service consumer must support the respective object model in addition to the messaging format. For example, a CORBA server application will be unable to serve a DCOM client and vice versa. This is an inherent drawback of OOT based distributed systems. It has become evident with the popularity of SOT that it is harder to standardize objects and the communication protocol among them than it is to standardize just the messaging format among arbitrary entities. A common object model leads to tight coupling, which cannot be enforced across administrative/organizational domains.

2.1.3 Jini and Salutation

Jini is a Java-based coordination framework derived from the Linda Tuple Spaces coordination model. A Jini federation is a set of independent devices that can discover and interact with each other using SOT. Jini provides a lookup service for various devices to find one another. However, it should be noted that Jini implicitly uses the Java object model. With this object model, objects on one device can use Java Remote Method Invocation (RMI) to invoke methods on remote objects residing on other devices. Each type of device is restricted to a highly-specified interface which enables dynamic interactions [4].

Salutation is device-coordination framework in which the bulk of the communication is managed by Salutation Managers (SLMs). SLMs interact with one another to provide coordination among devices. A service provided by a device is understood to be a set of Functional Units, and each Functional Unit provides a facility. A device that intends to interact with other devices needs to initiate contact with a local or nearby SLM to search for and invoke services. Each Functional Unit has well-defined specifications. Salutation-enabled devices are coordinated exclusively based on the services they offer to each other and the services they consume [4].

In terms of SOAs, Salutation provides the most loosely-coupled, open specification. However, both Jini and Salutation require standardized interfaces for services provided to enable interoperability. In addition, both are targeted for facilitating local coordination.

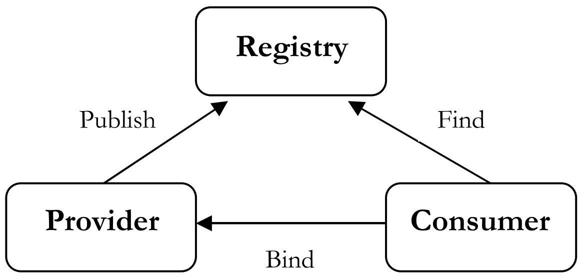
2.1.4 XML Web Services

Web Services is a collection of related technologies that, when considered together, provide a distributed computing platform. XML was the driving technology behind Web Services as it became the preferred data-exchange format. From these beginnings, Web Services emphasized the standardization of the communication protocols for uniform data exchange over the Internet. From a historical perspective the first XML-based implementations, such as XML-RPC [32], were initiated just two years after CORBA incorporated IIOP. Early adopters of CORBA, frustrated about its interoperability woes, now looked to Web Services instead for a solution.

The Simple Object-Access Protocol (SOAP), which was a successor of XML-RPC, became the messaging protocol between entities. It cannot be denied that wide-spread agreement was instrumental in the adoption of SOAP as the standard for Web-Services communication. However, SOAP does provide a number of useful capabilities that made achieving conformity easier. SOAP was designed not only to operate via universally-accepted Internet protocols, HTTP and SMTP, but also to be extensible to use proprietary protocols [33]. Communication among entities across administrative domains needed to take into account crossing firewalls. Web Services adopted SOAP over HTTP as the preferred method since it provided entities with a common port for interactions. Previous distributed computing platforms required workarounds that were intricate to implement.

To be fully operational as a distributed computing platform, Web Services adopted Universal Description Discovery and Integration (UDDI) [19] as a directory for looking up service-provision entities. Web Services Description Language (WSDL) [34], another XML-based technology, was used to describe the operations of services listed in the UDDI.

Web Services has been described using the “publish-find-bind” paradigm. Figure 3 shows how the various technologies described earlier fit into this paradigm. Though UDDI and WSDL are non-essential to developing Web Services, they provide for a more loosely-coupled interaction among entities that have not previously interacted. They allow the implementation of the service providers to be decoupled from that of the service consumers. Recent initiatives such as the Web Services Interoperability (WS-I) [31] have been geared to making Web Services technology truly independent of programming language, distributed component model, and operating system. Given these attributes Web Services provides for the most open and loosely-coupled SOA for distributed computing.



Service Provider publishes a description of the service to the Service Registry. The Service Consumer then finds the service by viewing its description in the Registry. The Consumer can then bind with the Provider of the service.

Service Registry UDDI
Service Description WSDL
Data Encoding XML
Communication HTTP/SMTP

Figure 2.3 Web Services – the Publish-Find-Bind paradigm and Technology Stack

2.2 Publish-Subscribe Systems

Every event-driven system relies on some sort of publish-subscribe mechanism. Publish-subscribe systems provide a highly scalable loosely-coupled interaction model. These systems are attractive for event-driven dynamic interactions because they can decouple service providers from consumers on the basis of time, space, and even operation flow [9]. Given these advantages and the context of the previous section, publish-subscribe technologies can be incorporated into SOAs for distributed computing. Here we just provide a brief overview of this research field.

Publish-subscribe systems can be classified according to various characteristics but the most popular differentiation is whether a system is subject-based or content-based. In subject-based systems publishers are classified according to a certain subject, usually encompassing distinct data about the subject. Content-based systems are of a finer granularity where each message is filtered by the publisher based on how closely it matches with the query of the subscriber. Other mechanisms to distinguish among publish-subscribe systems include their architecture, delivery mechanisms, reliability, security, and algorithms used to map a publisher's events with a subscriber's query [9].

Research has been geared towards developing a highly-scalable publish-subscribe middleware for public networks [9, 13]. Highly scalable refers to the amount of data that can be channeled from publisher to subscriber as well as the number of subscribers that can be accommodated. Public networks are classified as ones that use universally-adopted protocols for communication (i.e. TCP/IP, HTTP). In addition, researchers are constructing a publish-subscribe platform for supporting both subject and content based subscriptions [9]. A prime example of research in publish-subscribe systems is the Gryphon project [13].

From a research perspective, EDRA is not geared to addressing the problems of supporting a highly scalable platform for public publish-subscribe systems. While, EDRA could possibly make use of technologies developed in a project such as Gryphon but that is beyond the scope of this thesis. EDRA's focus is in automating the selection of publishers on behalf of the subscriber to acquire the appropriate subject or content information necessary for a logical response to changes in the subscriber's operating environments.

2.3 Semantic Web

The growth in the amount of data posted on the Internet has made locating all the relevant information a daunting task. Even Google, the most popular search engine returns a significant number of false positives while omitting many correct answers. The research efforts behind the Semantic Web aim to simplify this process to the level of enabling machines to search and process information seamlessly. Assessing relevance is one of the capabilities required for automatic establishment of service relationships. Therefore, we provide a brief background discussion on the Semantic Web and the semantic approach to assessing relevance.

The Semantic Web initiative first wants to redefine publishing on the Internet in a manner that enables reprocessing of the data as opposed to layout like HTML. Information sources are to be addressable through a unique Uniform Resource Identifier (URI), similarly to how users access web sites using Uniform Resource Locators (URLs). However, the significant difference is how various URIs are interconnected. In the traditional Internet the links are usually one-way, but the Semantic Web proposes multi-directional links among URIs that construct a virtual mesh of related information in the Internet [25].

To provide "semantic" association among information sources an XML-based language, called the Resource Description Framework (RDF), has been proposed. Each information source must be documented using RDF to simplify the searching of related information. The benefits of using RDF lie in the fact that it standardizes how distributed information can be associated with one another. The catch is that for the Semantic Web to be successful people must begin publishing using RDF on the Internet [25].

The next step is the use of RDF to establish a data-typing model or grammar for specifying information. This is achieved through the use of RDF Schema. The process is analogous to that of developing an XML Schema for processing of various XML documents that correspond to that schema.

DAML is an ontology and inference language based on RDF [8]. With a multitude of entities having created different RDF Schemas there was a requirement for tying them together, and the DARPA Agent Markup Language (DAML) filled that role. An ontology is defined as a specification of a conceptualization. For example, an ontology can be defined for the concept of an automobile (it consists of wheels, a chassis, *etc.*). Association of RDF types specified in different RDF schemata allows for the construction of ontologies. Ontologies are then used to make logical inferences using logic rules [25].

2.4 Related Research

The previous sections dealt with covering the various research areas that are pertinent to EDRA. In this section we highlight some of the research efforts currently being pursued or that have been completed to provide the research context for EDRA. The projects discussed here reflect the research directions in the research areas discussed earlier.

2.4.1 Semantic Calendars – RCAL

Semantic Calendars are an AI approach to solving the problem of scheduling meetings among a number of entities. Restina Semantic Calendar Agent or RCal [26] is one such project that aims to automate the selection of a mutually-convenient time for a meeting. An important feature of RCal is its use of AI agents that participate in "negotiating" the appropriate time for a meeting. Each entity's current schedule, priorities, and preferences are known to their RCal agent. RCal models the problem as a set of distributed information sources that collaborate to make the optimal resource allocation given a set of constraints. The resource being allocated is time and the information sources represent the RCal agents acting on behalf of the entities which want to schedule a meeting. RCal operates in parallel with calendar clients (i.e. Microsoft Outlook) and utilizes them as databases for storing calendar entries. RCal Agents maintain facilities to query calendar entries and negotiate with other entities to schedule meetings. RCal provides semantic annotations of user calendar entries and is interoperable with static RDF based information sources (Semantic Web).

At present, RCal exclusively concentrates on establishing new calendar entries for the user. RCal does not provide facilities to track events that might force changes in a calendar. Established meetings have to be manually undone and scheduling has to be reinitiated. 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. For example, a meeting can be scheduled for a certain time because the user had not previously specified constraints. However, if the time between when the meeting is scheduled and when the meeting should take place is relatively short then the context of the user will determine whether he/she will be able to attend.

2.4.2 Semantic Approach to Service Lookup and Integration

Paar and Tichy [24] 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. The developers' efforts are also simplified by incorporating additional information about the operations of a service rather then just syntactic names that are currently provided in WSDL files. Development of semantic web services involves applying semantic annotations to the source code that implements services, and WSDL files that describe the service. Semantic annotation is done using DAML+OIL as the markup language. DAML is also used for the development of ontological mappings. An additional feature the authors mention is the use of natural-language (i.e. English) phrases to supplement the semantic annotations. This enables users to interact with and select services to use during runtime. Given dynamic parameters such as location and the task to be achieved, a logical deduction process is used to select and invoke the appropriate service to address the problem.

This semantic-annotation approach to enable run time selection of appropriate services demonstrates potential. 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. It also facilitated the execution of test cases before deployment. Introducing automatic establishment of service relationships during run time will not be reliable for critical applications without a mechanism to certify service providers. There is no way to guarantee that semantic annotations of services correspond to the functionality of the service. 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. This level of support, albeit with a lower degree of accuracy can be provided without semantic annotations using syntactic matches, à la Google. The client can then choose which of the returned services to invoke.

2.4.3 Context Service

Context Service [15] is a service-based approach for integrating context-awareness into applications. Hui *et al.* argued that context-awareness enhances an application's ability to adapt its functionality to meet the needs of the user and enhance the user's experience. Previously, to make applications context-aware, custom infrastructures were built. However, this limited the flexibility by restricting the number and sources of context information. In addition, all the context information has to be managed by the application. In this approach the context-service can be queried by various applications to retrieve the contextual information pertinent to that application. Applications that make use of the context-service can also delegate context-management responsibilities to it. The context service receives contextual information from third-party context sources. The architecture to build this context service is extensible to support heterogeneous sources and the cost of incorporating additional sources can be amortized among all the consumer applications. The context service's design calls for facilitating privacy control.

The context service is a robust and modular approach to designing context aware applications. Its aim is to just 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 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.

2.4.4 Services Platform for Context-Aware Computing

The Services Platform for Context-Aware Computing [7] is similar to the Context Service in terms of providing contextual information as a service to applications that require it. This research project differentiates itself by proposing to build context-gathering capabilities into the Web Architectures for Services Platform (WASP). WASP is an overarching platform where all service consumers and providers reside, with specific emphasis on service delivery taking place on 3G networks. Within this framework, the WASP platform is required to provide for presenting data from various context services in a uniform manner to applications that invoke them. The platform also has the flexibility to support various interaction models. Applications that are made WASP-compatible will gain access to context-awareness services through a subscription process. WASP manages the flow of context information. The WASP context-awareness facility interprets information from the various context sources and either stores this information until needed or channels it to the appropriate applications that have subscribed to this information. WASP can also couple raw context information with other service providers to assist the application. Applications must make use of a standardized subscription protocol to dynamically interact with context sources and other service providers.

This approach to providing context-awareness capability is scalable, and provides facilities for dynamic event-driven applications. WASP is focused on enabling context awareness in applications and providing a common platform upon which to interact with other applications or services. Responding or providing adaptive capability based on context information for applications is not one of the facilities provided by WASP. Similar to the Context Service, WASP does not deal with the gathering or management of future contexts.

2.4.5 iQueue Project

The iQueue Project [6] was geared to address the issues surrounding data composition in reactive applications. iQueue is a distributed system which manages the network placement of data source composers for an application to optimize computations. This is a framework that allows applications to create composer objects to amalgamate data from various sources. A data composer is responsible for collecting one type of information and producing a presentation of all the information. The operation of the composers in aggregating data from various sources is left to the application developers. The iQueue system maintains a set of member data services, and when functional data specifications are provided by an application to its composers they selects a subset of those services to use as data sources. Composers themselves may supply their aggregated information to other composers, creating a hierarchy. The system is also designed to reselect the data sources during run-time based on quality-of-service requirements of the client application.

iQueue is focused on optimizing aggregation of information from various sources. It concentrates 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.

2.5 Chapter Synopsis

In this chapter we provided an overview of the research background for the EDRA project. We then discussed the various research projects in these research areas to lay out the research context for EDRA. Through our discussions we outlined the distinguishing features of prior works and described how they differed from the goals of EDRA.

Chapter 3

Structural Design of EDRA

In Chapter 1, we provided a glimpse of EDRA framework. The intent of this chapter is to provide the complete structural layout. We will specify all the necessary components that together define EDRA so that every facet of the framework can be conceptualized. To facilitate understanding we identify the use of common design patterns. A discussion on the reasoning behind design choices of each structural component will also be provided. We simultaneously cover the essential interfaces of the various modules of design elements. Scenarios described in Chapter 1 will be used here again as illustrative examples. Overall the infrastructure required for our approach to addressing the general concepts introduced in Chapter 1 will be in place before we examine the procedural aspects of EDRA. Through this exercise we will be able to smoothly transition to the behavioral design of EDRA which will be discussed in the subsequent chapter.

3.1 Design Overview

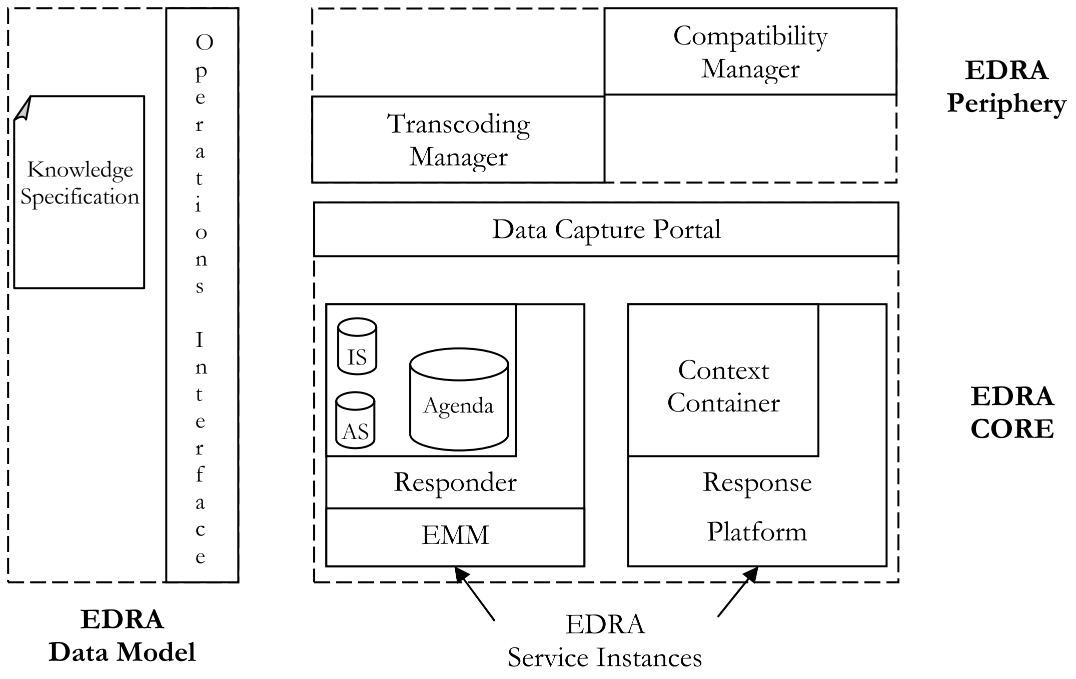


Figure 3.1 A more detailed view of the EDRA framework

The high-level design of EDRA was influenced by the Model-View-Controller design pattern. In MVC [23, 27], the *model* is responsible for maintaining and processing information that is relevant to the client of the system. The *view* provides clients with the ability to access the current state of the information in the model. Finally, the *controller* facilitates client manipulation of the information in the model. Each model may have multiple views and controllers. The MVC design pattern was adopted in the design mainly because of the diversity of access the EDRA framework provides the client. Recall from Chapter 1 that the main classes of design elements were the EDRA Periphery, EDRA Data Model, and EDRA Core. The EDRA Periphery provides the view and controller functionality for the EDRA framework. The EDRA Core behaves as the model for the framework. The EDRA Data Model plays a supporting role by defining properties of the information entering, contained in, and leaving the EDRA framework. Figure 3.1 provides a detailed illustration of the whole framework (see Figure 1.3).

The EDRA Periphery is necessary to manage the bulk of interactions with the client. Client interactions that deal with creating, updating, accessing, and removing EDRA service instances will be done via the EDRA Periphery. These types of interactions can occur either manually by the client or through client-invoked applications. For example, a client (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. EDRA Periphery is required to map information from clients and other external sources (e.g. Applications) to the data model of the client's application domain.

EDRA provides for loose coordination among dynamically-interacting entities by relying on 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. Each application domain will have abstractions that generate requirements on what the data is and what it means. For example, in a business process, data elements are used to describe operations (e.g. Ship: goods are moved from warehouses to stores) and generalizations (e.g. Supply chain: encompasses all operations involved in moving goods from the supplier to the customers). The EDRA Data Model puts forward that for each application domain the data can be standardized into a model. It provides an interface through which other components of the EDRA 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 client's application domain.

The EDRA Core is the heart of the EDRA framework. Operation facilities that the EDRA framework advertises are supported by the EDRA Core. The API of the EDRA framework that clients will make use of is essentially the API of the EDRA Core. 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 we have mentioned before, 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.

Subsequent sections of this chapter will dwell deeper into the details of the EDRA Periphery, EDRA Data Model, and EDRA Core.

3.2 EDRA Periphery

One of the generalized concepts EDRA intends to address is: One Point of Data Entry (see Section 1.2.1). It is the EDRA Periphery's responsibility to meet this goal since it is the view and controller of the EDRA framework for clients. The EDRA Periphery manages its responsibility through the Transcoding Manager and the Compatibility Manager. Together they establish the infrastructure capable of providing the client with one point of data entry functionality. The next chapter will attend to the behavior of this infrastructure to achieve the goal.

Clients can request a display of the current state of their EDRA service instance through an access device. An access device may be a cell phone, PDA, or a PC. The Transcoding Manager supports all display interactions between a client and the associated EDRA service instance that occur through an access device. In terms of the MVC, the Transcoding Manager plays the role of the view. Clients may intend to add, change, or remove information from their EDRA service instance. This can be achieved either through an access device where the client updates information in an EDRA service instance manually or through external applications. The Transcoding Manager handles update interactions that occur though an access device and the update interactions that occur through external applications are handled by the Compatibility Manger. In terms of the MVC both the Transcoding Manager and the Compatibility Manager serve the controller role.

3.2.1 Transcoding Manager

The Transcoding Manager, as its name suggests, transcodes information from the client's EDRA service instance into an appropriate format that is suitable for the access device being used by the client. For example, a client's use of a cell phone will place a number of restraints on displaying information due to its user interface, communication medium, and other considerations that would be different if the client used a PC. Since the transcoding algorithm will vary depending on the access device being used, the Transcoding Manager is constructed using the Strategy or Policy design pattern [10]. This design pattern enables us to encapsulate a family of algorithms that perform a similar task and be able to interchange among them based on the needs of client being served.

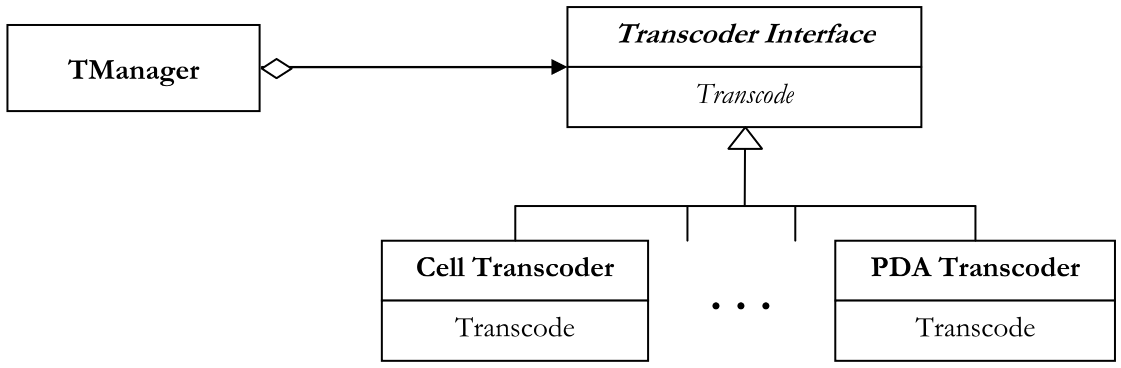


Figure 3.2 Transcoding Management with Strategy Design Pattern

Figure 3.2 shows the components that make up the Transcoding Manager: the TManager, Transcoder Interface, and implementations of the Transcoder Interface. The TManager maintains a reference to the Transcoder Interface. The Transcoder Interface defines all the common operations through which information will be appropriately transcoded. Each implementation of the Transcoder interface will be specific to an access device. Providing support for an additional access device, which may be used by the user would require providing a corresponding implementation of the Transcoder Interface. For the purposes of this project, we assume that the Transcoding Interface has only one operation – Transcode(data), where data is what needs to be transmitted to the client. The actual process of transcoding is its own research area and is beyond the scope of this thesis. The TManager can then be configured with a concrete implementation of the Transcoder Interface when a client initiates interaction. The TManager maintains a reference to the EDRA Core so that it can have direct access to the interface the EDRA Core provides.

Thus far we have only considered the flow of information from an EDRA service instance to the client when constructing the Transcoding Manager. Since the Transcoding Manager must also behave as a controller (in terms of the MVC design pattern), information needs to flow from the client into an EDRA service instance. To address this issue along with maintaining compatibility between information the client offers and the EDRA Data Model, the Transcoding Manager provides an explicit configuration for the input mechanism. For now, it will suffice to assert that the structure of the Transcoding Manager presented is also capable of managing the flow of information from the client to an EDRA service instance. In the next chapter we will substantiate this assertion when discussing the behavioral design.

3.2.1.1 Transcoding Manager Interface

The Transcoding Manager's interface with the EDRA Core is outlined below. How the Transcoding Manager interfaces with the client falls under the behavioral design and will be left for Chapter 4.

- TManager(EDRA_Core) – creates a new TManager upon receiving a service connection request from a client with a reference to the EDRA Core.
- Set_Transcoding_Alg(Transcoding_Interface_Impl) – identifies which implementation of the Transcoding Interface that should be used when interacting with the client.
- Start_Session() – begins the interaction between the TManager and the client.
- Close() – closes the service connection with the user and ends the TManager instance dealing with that connection.

3.2.2 Compatibility Manager

The Compatibilty Manager assures that all information flowing from an external application into the EDRA framework is compatible with the EDRA Data Model. This requires adapting the data output from external applications so that it can be mapped to the type, structure and representation of data as prescribed by the EDRA Data Model for that application domain. For example, a client may use a travel booking application that creates a trip schedule that needs to be imported into the client's EDRA instance. The data format of the custom travel application needs to be mapped to the Travel EDRA Data Model so that the EDRA Core operations can process the data appropriately. Therefore, the Adapter design pattern [10] is used to construct the Compatibility Manager. This design pattern enables us to encapsulate the mapping of data between an external application and the EDRA framework into an adapter component. This is an implicit assumption that the adapters are cognizant of the appropriate EDRA Data Model.

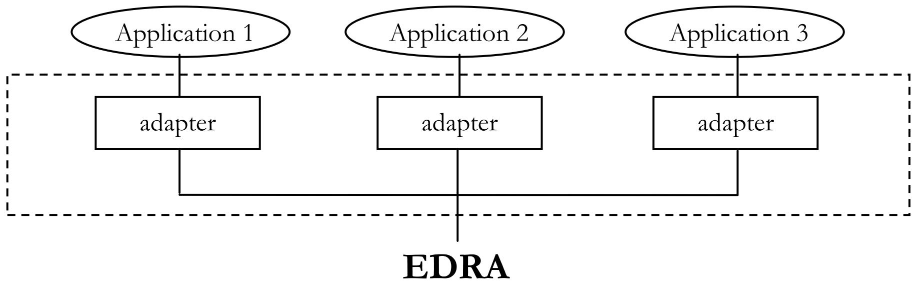


Figure 3.3 Compatibility Management using the Adapter Design Pattern

The Compatibility Manager is realized through a collection of adapter components, which are depicted as the dashed rectangle in Figure 3.3. Each external application would require a unique adapter component. All the adapter components will make use of the common interface the EDRA Core provides, but each will maintain unique interfaces with the specific external application they provide support for. Clients are required to register all the applications they intend to transfer information from to their EDRA service instances prior to actually using them. This is a valid assumption since there are a limited number of applications that a client may use regularly to plan current and future contexts. To allow new external applications to interoperate with the EDRA framework, a corresponding adapter component must be integrated with the application and alert their EDRA service instance of the new application.

An external application may then require information stored in a client's EDRA service instance. The EDRA Core's interface with an adapter will permit information flow in both directions. However, the interface specification between the adapter and the external application will determine whether the application can import information from an EDRA service instance or not.

3.2.2.1 Interface of Adapters

We now document the interface of an adapter with the EDRA Core.

- Connect(Client_Reg_Info) – connects to EDRA Core and accesses the EDRA service instance of the a client identified by the Client_Reg_Info.
- Import(Request) – retrieve information based on a request from the EDRA service instance.
- Export(Information) – transfers information to the EDRA service instance in an EDRA Data Model compatible format.
- Disconnect() – disconnects from an EDRA service instance.

3.3 EDRA Data Model: Industry Vertical Type Systems (IVTS)

The EDRA Data Model is central for loosely-coupled, service-based coordination between the EDRA framework and external services and applications. We will begin by motivating our approach and then present the infrastructure we propose to construct it.

3.3.1 Motivating Trend

We have stated earlier that the EDRA framework aims to achieve coordination through the standardization of data. The EDRA Data Model is the approach taken to achieving it. To corroborate this approach it is important to observe industrial trends with respect to integration of IT systems within and without organizations.

One of the main hurdles in IT systems integration is conflicting or incompatible data formats [16]. To overcome this problem initiatives are being taken within industries (e.g. Healthcare, Banking) and within organizations (e.g. Human Resources) to achieve standardization on data. Each industry or our preferred term, application domain, maintains a distinct vocabulary, defines suitable abstractions, and outlines various associations among these abstractions. Data within an application domain or industry is typed, structured and represented based on this common understanding. This would result in a standardization of data.

Previously, each IT solution adopted a data format individually, creating conflicts and incompatibility [14]. More recently, task forces and committees in various industries and in organizations (inter-department) have been established to formalize vocabulary, entity specifications and relationship definitions. For example, Health Level Seven [12] and Open Financial Exchange [22] are initiatives by the healthcare management industry and the banking/financial industry respectively to achieve standardization of data interchange formats across IT solutions. On a much smaller scale a number of XML-Schema definitions are being made available publicly to enhance collaboration for certain tasks [18]. Also, recall from Chapter 2, the Semantic Web approaches standardization of data through the use of ontologies.

We have motivated the idea for an EDRA Data Model from the trend towards collaborative standardization of data. We expect that this trend will continue and hence make our approach to achieving coordination in service-based computing a plausible one.

3.3.2 EDRA Data Model Infrastructure

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. This forms the foundation for addressing the generalized concept: Robust Architecture Customized for Various Applications (see Section 1.2.2). We are able to use the same EDRA framework for varied application domains since the main difference among application domains is their data. We separated the EDRA Data Model from the rest of the EDRA framework so that we can replace one IVTS with another based on application domain we want to cater to.

IVTS is best explained by discussing its two components the knowledge specification and the EDRA operations interface separately. The reason behind separating the two components will become evident when we discuss possible deployment options of the IVTS. Figure 3.4 captures the essential parts of the IVTS.

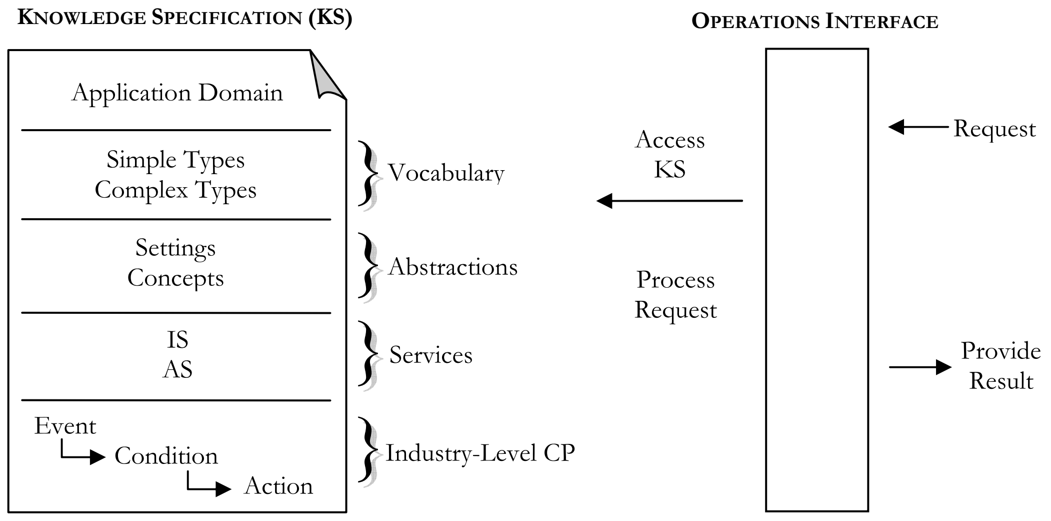


Figure 3.4 Components of the EDRA Data Model

3.3.2.1 Knowledge Specification

Knowledge specification has multiple purposes, but the most significant is the standardization of data within the application domain. Though an application domain is not constrained in the approach it takes to standardize data we suggest certain constructs that have to be incorporated for the EDRA framework to operate as we envision it. The constructs we have suggested were derived either from current initiatives in standardizing data or through the needs we identified in our scenario-based development process.

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. Vocabulary is defined using basic types supported by specification languages such as integers, strings, booleans, etc. This constitutes the extent of most data-standardization efforts today (e.g. XML schemas). 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. Entries may be either Settings or Concepts. A Setting denotes the discrete operating environment for an abstract activity. Recall that an operating environment is the set of properties that constitute a client’s context within an application domain. A Concept is a generalization that is used to categorize one or more Settings and/or one or more Concepts based on procedural relationships. This is best illustrated using an example from our travel scenario. A client’s travel plans could be identified as the Trip Concept and each phase of the trip, like the flight, meeting, etc. could be identified as a Setting. We can relate this to formalizing the representation of information for input into a database. In this analogy, we can map Settings to entities and Concepts to relationships. When creating the knowledge specification, links among various Settings and Concepts may become apparent resulting in it becoming a hierarchy or a mesh. When clients enter their information into an EDRA service instance (see Figure 1.3) it is stored as a collection of Entries. This segment of our data standardization will vary in complexity depending upon the application domain and may include other constructs not detailed here.

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 (e.g. Web Services) within which the EDRA framework is deployed. 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. Here again we make the suggestion that all services be classified into two groups, information services, and action services.

Information Services (IS) are subscription services that provide update information about specific operating environments when changes occur. IS can provide contextual information about clients or their operating environments and these are the services automatically subscribed to when clients enter information into their EDRA service instance. Since each IS service maintains a set of entries for which the service provides relevant updates, the EDRA Core can compute the exact collection of IS services to subscribe to based on the client's set of Entries. An example of an IS service in the business process example would be a service that provides updates on the inventory levels in the warehouse. If we assume that "Supply Chain" is a Concept within the business process application domain and Stocking the warehouses as a Setting then both would be referenced by the warehouse inventory update service as Entries for which it is relevant.

Action Services (AS) on the other hand can be invoked to change the client's current or future operating environment. An AS service can be provided to the client when semi-automating a response to a change in the client's current or future operating environment. A hotel booking service would be an example of an AS service. Any external application that can provide its functionality as a service using a Compatibility Manager adapter can be an AS service.

The knowledge specification can provide additional resources to identify relevance between services and Entries by naming service providers and levels of quality of service and other classifying terms. Services, both IS and AS, can then be further classified.

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. Contingency policies are essentially a set of event-condition-action rules. Upon receipt of an event, a condition is evaluated, and based on the result of the condition an action is taken. EDRA also allows Entry-level contingency policies but this is to be specified by the client whereas the industry-level contingency policies are industry defaults used when the clients prefer not to define their own.

The knowledge specification is unrestricted in the types of information it would want to document and standardize based on its pertinence to the application domain. The suggestions that have been made are not set in stone but it should be noted that the knowledge specification will significantly influence processing operations of the EDRA Core. Therefore, any additions that are made to the knowledge specification must also include updates to the EDRA Core processing operations and any modifications to the suggested outline of the knowledge specification will require parallel modifications of the EDRA Core.

3.3.2.1.1 Entries, Settings, Concepts

Entries, Settings, and Concepts can be conceptualized as a tuples, which contain the content necessary to describe operating environments within an application domain. There exists an inheritance relationship between Entry, Setting, and Concept. Settings and Concepts extend the Entry tuple by adding additional content. The inheritance hierarchy does not extend beyond Setting and Concept. Specific Settings for an application domain contain a sequence of variable content that is appended to the original tuple for a Setting. A specific Concept is realized similarly.

We characterize the Entry tuple as follows:

- Name – the name of the Entry.

- Start_Time – the time at which the client enters the operating environment which this Entry represents.
- End_Time – the time at which the client leaves the operating environment which this Entry represents.
- Duration – the length of time the client will spend in the operating environment which this Entry represents.
- Default_CP – a default contingency policy to be executed if and when an event is directed to this Entry and no specific contingency policy exists to respond to the event.
- List of Client_CP – a list of contingency policies specified by the client for a number of events that may be directed to this Entry. (Entry-level contingency policies)

We characterize the Setting tuple as follows:

- Description – a description of the operating environment this Setting represents.
- Exclusive – a boolean value that specifies whether the Setting is the exclusive context of the client. At any point in time at most one exclusive Setting defines the operating environment but it may overlap with one or more non-exclusive Settings.
- Priority – a mechanism used for ordering Settings of a client other than time.
- Variable_Content – content specific to a certain Setting (e.g. Flight Setting would have Flight Number, Seat Number, etc.).
- Elements of the Entry tuple.

We characterize the Concept tuple as follows:

- Description – a description of the operating environment this Concept represents.
- Recommended_Combinations – outlines the various combinations of other Concepts and Settings that can be grouped into this Concept.
- Variable_Content – content specific to a certain Concept (e.g. Trip Concept will have mode of principal transport, type of trip, etc.).
- Elements of the Entry tuple.

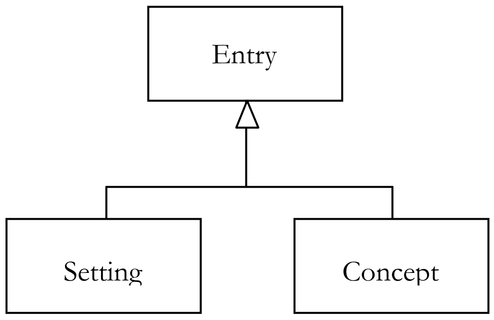


Figure 3.5 Entry inheritance hierarchy

3.3.2.2 Operations Interface

The second component of the IVTS is the operations interface. The knowledge specification is generic to the extent that it will be posted publicly for all external services that want to conform to the IVTS of an application domain. The operations interface is more focused on providing facilities for EDRA Core to interact with the knowledge specification of the IVTS. Facilities this interface provides may be suitable for other services other than the EDRA Core. For this reason it remains separate from the knowledge specification because it adds flexibility in deployment. The operations interface can be hosted along with the EDRA Core and be used exclusively by EDRA or it can be hosted publicly as a service.

- Get_Industry() – identifies the application domain of the IVTS knowledge specification uniquely.
- Enumerate_Entries() – lists all the Entries (e.g. name & description) documented in the IVTS knowledge specification.
- Search(Entry) – searches the IVTS knowledge specification for an Entry’s specification.
- Search(Phrase) – searches the IVTS knowledge specification for an Entry based on the Phrase. Exactly how Phrases are used is left to the implementation of the Search algorithm. For example, Phrase could be used as a keyword in the search.
- Get_IS(Entry, Classifiers) – retrieves all the IS services relevant to an Entry. Classifiers can be used for selecting among relevant IS services.
- Get_AS(Entry, Classifiers) – retrieves all the AS services relevant to an Entry. Classifiers can be used for selecting among relevant AS services.
- Get_CP(Entry, Event) – retrieves the industry-level contingency policy for an affected Entry based on the Event.
- Get_Ref_IVTS() – retrieves the URI of the IVTS knowledge specification.

3.3.3 IVTS and Service Discovery – Design/Deployment Note

The EDRA framework is geared to operate within a service-based computing environment. Most SOA solutions, such as Web Services, provide their own service repositories. It might seem redundant that the IVTS knowledge specification provides a listing of all available services that can interoperate with the EDRA framework. However, IVTS can leverage existing service repositories to achieve the same functionality.

The SOA solution-specific service repository (e.g. UDDI) would contain pointers to EDRA compatible external services. The IVTS knowledge specification, instead of keeping track of the services would be used in defining the search of the service repository to find appropriate services. Within the knowledge specification, service categories would be specified for sets of Entries (as opposed to the services themselves), and then these would be further categorized based on other classifying terms. Using this information the service repository can be searched. For example, a client needs flight information which is part of category F1, but the flight is of a specific airline whose category is A5. The intersection of both categories will yield the most relevant service.

This level of indirection is an advantage because it allows us to address the generalized concept: Certifying Services Employed (see Section 1.2.5). We had mentioned earlier that initiating service relationships automatically carries reliability and dependability risks. To overcome this we can establish a certification process for service providers to include their service into an IVTS's knowledge specification. Only services which satisfy the claims made by the service provider will be certified. Only certified services can be included within IVTS defined categories. If so required, a new category can be incorporated for a new certified service. In a heterogeneous application domain with numerous service providers maintaining dependability and reliability can be tackled using a community-based rating system. Service consumers can rate certain services poorly if they are dissatisfied and over a period of time these poorly maintained services will not be used by consumers when alternative services are available.

3.4 EDRA Core

The EDRA framework can be deployed to serve multiple clients or just one. We will discuss the more complex alternative of serving multiple clients and by doing so cover all the requirements of the simpler case.

EDRA Core's high-level design is based on the Façade design pattern [10]. The design requirement was to provide a unified interface for external access to EDRA Core's features. Recall that the EDRA Core has three subclasses of design elements. Data Capture Portal behaves as the façade for the other subclasses of EDRA Core. Instances of the Context Container (Storage) and the Response Platform are paired for creating EDRA service instances for clients. Both subclasses of design elements provide an interface through which they interact with the Data Capture Portal. The next three subsections will layout the specifics of the Data Capture Portal, Context Container and the Response Platform.

The EDRA Core carries much of the responsibility for addressing the generalized concepts: Managing Present and Future Client Context (see Section 1.2.3), and Client Control of Client Context (see Section 1.2.4). Also, it needs to assist the EDRA Data Model to completely address the generalized concept: Robust Architecture Customized for Various Applications (see Section 1.2.3 and Section 3.3.2). The Context Container and the Response Platform share the responsibility for managing the clients' contexts for a pre-defined period of time. Its position in the EDRA Core makes the Data Capture Portal capable of authenticating access to a client's EDRA service instance. This is one step towards assisting the client to be in control of their current and future contexts. The Context Container adds to the support provided by EDRA Data Model so that various scenarios within an application domain can be managed by the EDRA framework.

3.4.1 Data Capture Portal

The Data Capture Portal is designed using the Singleton design pattern [10]. This is to guarantee that the EDRA Core has one global access point. As the single access point, the Data Capture Portal interfaces with the EDRA Data Model through its operations interface, the TManager, and adapters of external applications. Apart from its most significant responsibility as the façade for the EDRA Core it handles the initiation of every EDRA service instance 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.

3.4.1.1 Data Capture Portal API

Given the extensiveness of the EDRA Core API as presented by the Data Capture Portal, we will cover it by just discussing the high-level facilities it provides. Attempting to mention all the operations of the API will force us into implementation level details shifting our focus away from the structural design.

The first facility that the Data Capture Portal provides is the ability for clients to establish connections to engage in information exchange with the EDRA Core. The initial connection made by the client will be to register for an EDRA service instance. The necessary operations for the registration process can be categorized as another facility. Subsequent connections made by the client to the EDRA Core will be to add, update or remove Entries into their EDRA service instance. This will also involve clients adding, updating, and removing dependencies and Entry-level contingency policies for Entries they have in their EDRA service instance. Operations that enable these actions constitute the third facility provided by the Data Capture Portal. Another facility exists to enable a client to import data into or export data from his/her EDRA service instance. The main use of this facility will be to move data in bulk. The last facility currently supported by the Data Capture Portal allows clients, when connected, to view and execute response recommendations for changes in their current or future contexts. The response recommendations we refer to, will have been previously sent to the clients by the Response Platform associated with the client's EDRA service instance.

3.4.2 Context Container

The Context Container is the warehouse of client related information as shown in Figure 3.6. 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, list of services that are relevant to the client, and pre-defined policies for responding to changes. The Context Container can be thought of as the client's personal database for that specific EDRA service instance. In terms of the MVC design pattern, the Context Container fulfills the data storage role of the model.

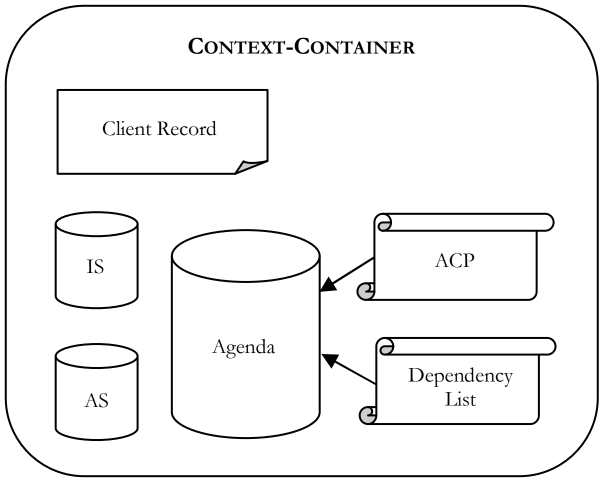


Figure 3.6 Visualization of the Context Container

A subcomponent, called the Client Record, maintains the client registration information. Registration information can include the name of the owner of this EDRA service instance, and contact information, especially the preferred type for notification (e.g. Email, phone call, etc).

The most critical subcomponent is the Agenda. The Agenda maintains the client's current and future context. Remember that 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 as shown in Figure 3.7. Dependencies among these Entries are captured by the Agenda's Dependency List, forming a partial order. Agenda can be interpreted as a schedule for the client. This is important 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 IS service. The client also has the flexibility to define appropriate responses at the Agenda level, possibly import corporate contingency policies. The process that determines which response is actually executed will be discussed in Chapter 4.

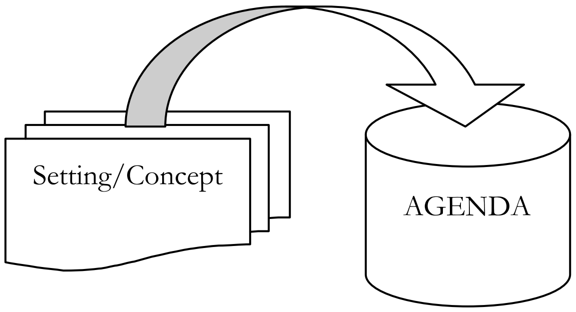


Figure 3.7 A client's schedule

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

3.4.2.1 Context Container Interface

The Context Container's interface is a culmination of all the interfaces of the various subcomponents. The core operations of all the subcomponents are the access, update, and removal of the data they store. The interface directed towards the Client Record allows manipulation of the client's Name, preferred mode of contact, registration identifier and password. The interface defined for the Agenda allows manipulation of the client's Entries, dependencies among Entries and the Agenda-level contingency policies. The interface for the Service Repositories allows manipulation of the services stored in them and references of relevance from those services to Entries in the Agenda. The Services Repositories also have search facilities to find services with references to a given Entry in the Agenda.

3.4.3 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 IS services. The Response Platform is also the infrastructure required to semi-automate client responses to notifications received from subscribed IS services by identifying the relevant AS services. All processing of the EDRA framework on the data stored in the Context Container is achieved through the Response Platform. The Response Platform serves as the information processing half of the model in terms of the MVC design pattern. The Context Container and the Response Platform together form the infrastructure for managing a client's current and future contexts.

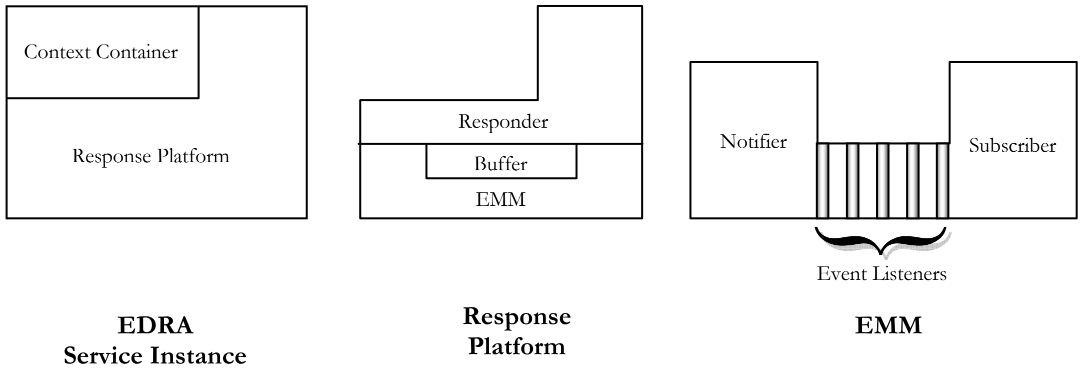


Figure 3.8 Response Platform and its constituents

The roles of the Response Platform are divided among the Responder, the Event Management Module (EMM) and the Buffer component. Figure 3.8 provides a pictorial representation of the Response Platform. The Responder is the driver of the Response Platform. It maintains a reference to its associated Context Container, a unique EMM and a unique Buffer. Interactions among the Context Container, Responder, and EMM were designed using the Mediator design pattern [10]. The Responder mediates the interaction between the Context Container and the EMM. We employ this design pattern to maintain loose-coupling between the data storage and update data retrieval. We encapsulate the processing needed to merge the data in the Responder. The EMM, when directed by the Responder, is responsible for establishing dynamic transient service relationships with external IS services. EMM manages the subscriptions and funnels the event notifications received from these services to the Buffer. The Responder retrieves events from the Buffer and semi-automates a response to the event if necessary. The Buffer allows EMM to get event information to the Responder without having to maintain a reference to it.

EMM contains three subcomponents, the Subscriber, Notifier, and Event Listener. The Subscriber manages subscriptions to IS services. Our reason for the Subscriber to be a separate subcomponent is because in the future we envision it being responsible for negotiating service contracts including cost for using services. With each new subscription the Subscriber creates an Event Listener that will receive updates from that service. The Subscriber will remove Event Listeners when the service contract expires or when directed to do so by the Responder. Event Listeners populate the Buffer when events are received. The Notifier is the subcomponent responsible for informing the client of the response, if any, taken by the Responder.

The Responder keeps track of events by checking the Buffer. An event would require the Responder to determine the change in an operating environment of the client and respond based on the contingency policies in the Context Container. The response taken, suggested response or simple client notification that the Responder initiates will be stored in the Responder's recommendation list and given to the Notifier to transmit to the client. The behavior of the Responder will be revisited in Chapter 4.

3.4.3.1 Response Platform Interface

In laying out the interface of the Response Platform we must actually layout the interfaces of the components that form the Response Platform. We begin with the Interface of the Responder.

- Responder(Context_Container) – a new Responder instance is by associating it with a Context Container instance.
- List_Previous_Recommendations() – lists all the responses taken by the Responder and all notifications of recommendations sent to the client.
- Respond() – responds to events that are added to the Buffer by detecting changes in the client's operating environment and using contingency policies.
- Populate_IS() – subscribes to IS services that are relevant to the Entries in the Agenda of the Context Container and adds successful subscriptions to the IS Repository in the Context Container. The IS services are initially retrieved from the IVTS
- Populate_AS() – finds relevant AS services from the IVTS and store them in the AS Repository.

The EMM interface is a collection of the interfaces of its subcomponents. We first present the operations of EMM interface pertinent to the Subscriber subcomponent.

- Subscribe(IS_Service) – creates a subscription on behalf of the client to an IS service.
- Unsubscribe(IS_Service) – removes a subscription to an IS service.
- Negotiate(QOS, Payment, …) – future functionality; negotiation for level of service and payment interactions.

The EMM operations pertinent to the Notifier subcomponent are provided next.

- Notify(Change) – clients are notified of changes in an Entry in their Agenda if no contingency policy other than the send notification default was specified for the Change.
- Notify(Response) – clients are sent the response taken about a change that occurred in Entries in their Agenda after following contingency policies.
- Notify(Suggestion) – clients are sent recommendations and AS services to use to modify Entries in their Agenda due to changes.

The Event Listener subcomponents are created dynamically to interface with external IS services. They will invoke these services using dynamic invocation interfaces provided by the SOA.

The Buffer provides an interface that is used the Event Listeners to send information about events to the Responder. Buffer can be thought of as a message queue between Event Listeners and the Responder.

3.5 Chapter Synopsis

This chapter put forth our infrastructure for addressing the generalized concepts introduced in Chapter 1. We presented the structural design of the EDRA framework by providing a detailed discussion on design choices, layout of the components and interrelationship among the components. Interfaces through which EDRA components interact among each other and external sources have also been presented. The necessary components we described for building the EDRA framework were derived through our scenario based development process. This will become more evident as we explain the manner in which these structural components work together. Chapter 4 will describe the behavior of these structural elements in fulfilling the objectives of the EDRA framework.

Chapter 4

Behavioral Design of EDRA

The purpose of this chapter is to identify and discuss the set of behaviors the EDRA framework defines. These behaviors are the algorithms that govern how the structural components discussed in the previous chapter operate and interact to provide a cohesive solution. An EDRA solution is the deployment of EDRA for an application domain. First, we outline a typical case for which EDRA can be deployed. Since we are discussing the behavioral design of EDRA, a detailed scenario that can be referred to throughout the chapter to provide illustrative examples is necessary. The rest of the chapter will concentrate on presenting the behavioral aspects of EDRA.

4.1 The Case – A Typical Scenario

The case we present takes place in the healthcare application domain. Specifically it deals with management of in-patient care at a hospital. The choice of application domain was made for three reasons. The first is to demonstrate the applicability of EDRA in a variety of industries. Since we have already mentioned travel and business-process scenarios, here we depict a scenario from the healthcare application domain. Second, the healthcare scenario has similarities with both the travel scenario and the business-process scenario, which were the original motivations for EDRA. The manner in which current and future contexts are entered by the client (via an application or manually) is similar in both the healthcare domain and the travel domain and the prescribed schedules are frequently adapted to changing requirements. Similar to a business-process scenario, a healthcare situation has a higher degree of control over the deployment of an EDRA solution such as well-defined contingency policies, and a standardized common knowledge specification. The third reason is that healthcare scenarios have some unique features that encourage the deployment of EDRA. The healthcare application domain has abundance in variety and number of medical systems and applications which can expose their functionalities as services. These medical systems and applications are very specific to the patient's case making it easier to determine relevance of services to client contexts.

A healthcare specialist, such as a physician, is assigned new patients as and when they are admitted into the hospital. Each patient's healthcare specialist creates a schedule for the treatment after an initial assessment. As the patient progresses through the treatment schedule it may necessary to modify or update it. For example, a procedure may need to be repeated or the initial assessment may have been too cautious (healthcare institutions prefer to err on the side of caution) and certain procedures could be skipped, etc.

The example scenario we use is about John who suffered a knee injury playing soccer and is unable to move his leg. When he is admitted into the hospital, Nancy, a healthcare specialist is assigned to him. Nancy makes an initial assessment and specifies the treatment schedule. John is prescribed non-inflammatory/pain killers and the dosage is included in the treatment schedule. John's treatment schedule is given below.

- 2pm – 5pm: First dosage of anti-inflammatory/pain killers. (No other medication for the next three hours)
- 2:30pm – 3pm: X-Ray of knee.
- 3pm – 4pm: Heat/Cold treatment.
- 4pm – 5pm: Rest.
- 5pm – 5:30: Preparation for surgery – anesthesia.
- 5:30pm – 9pm: Surgery.
- 9pm – 9am: Rest and dosages of anti-inflammatory/pain killers.
- One day: Monitor recovery and dosages of anti-inflammatory/pain killers.
- 9am – 9:30am – X-Ray of knee.
- 9:30am – 10:30 am: Apply an initial cast to the knee.

This schedule is an anticipated schedule for John and may be changed by Nancy as he progresses through his treatment. For example, he may not need surgery if the first X-Ray shows that it is just a bruise and not torn ligaments. Given this case, we now transition to understanding the behaviors of EDRA that establish how the EDRA solution would facilitate inpatient care at healthcare institutions.

4.2 Behavioral Design Overview

To begin with, the behavioral design of the EDRA solution has been based on it being deployed as a hosted service. EDRA Core will reside on a server with clients using access devices (e.g. PCs, PDAs, cell phones, etc.) to access their EDRA service instance. We do not require access devices to have any special type of support for EDRA, so long as they are compatible with the SOA middleware (e.g. Web browsers to access Web Services) used in the implementation.

A high-level view of EDRA's behavioral design reveals a three-step process as depicted by Figure 4.1. The initial step is the acquisition of a client's current and future contexts. These contexts would identify the operating environments of the client over a period of time. During the second step, this context information is used to determine relevant subscription services that are capable of providing update notifications when changes take place in the operating environments of the client. The final step involves formulating and executing a response to change notifications through a semi-automated reconfiguration of clients' current and future contexts. This process can be iterative.

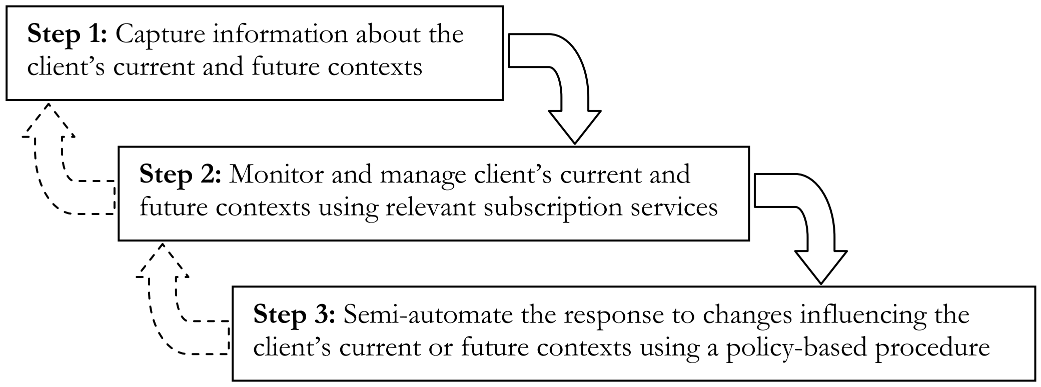

Step 1: Capture information about the client's current and future contexts
Step 2: Monitor and manage client's current and future contexts using relevant subscription services
Step 3: Semi-automate the response to changes influencing the client's current or future contexts using a policy-based procedure

Figure 4.1 High-level behavioral design of EDRA

The EDRA solution deployed in the healthcare application domain can model either patients or healthcare specialists who look after patients as the client. In our examples we will use the patient-centric approach. In terms of John's hospital scenario, the treatment schedule outlined for him by Nancy would be the acquisition of John's current and future contexts while he is at the hospital. Nancy subscribing to services in the hospital that provide such information as the number of critical cases, demand for the radiology facilities, patient heart rate (by room) among others would be automated by EDRA during the second step of process. Finally, adapting John's treatment schedule to changes would be achieved during the third step. An increase in the number of patients requiring immediate attention could be a change that can delay John's treatment. Below we identify the core set of behaviors defined for EDRA that make the three-step process possible.

The main behaviors defined by the EDRA framework are listed below.

1. Access – how EDRA handles client initiated interactions.
2. Registration – how a client can create a new EDRA service instance.
3. Acquisition of Contexts – how current and future contexts of the client are initially acquired.
4. Initialization – how an EDRA service instance prepares for the Management & Monitoring behavior.
5. Monitoring and Management – how EDRA interacts with external services and tracks changes in the clients' current or future contexts.
6. Response/Adaptation – how EDRA semi-automates responses to changes it has received from external services.

The Access behavior has been designed to provide a non-intrusive and simplified information gathering method. In Chapter 1, we stated that this is one of the basic objectives of the EDRA framework to fulfill the requirements of the generalized concept: One Point of Data Entry (see Section 1.2.6). Registration is a required auxiliary procedure for addressing the generalized concept:

Client Control of Client Context. The Acquisition-of-Contexts behavior is an essential first step to meeting the goals of the generalized concept: Management of Present and Future Concepts. Initialization plays a key role in defining the mechanism for automating the selection and invocation of client-relevant services during runtime, another basic objective of the EDRA framework (see Section 1.2.6). The Monitoring and Management behavior concentrates on building a preemptive approach to identifying changes that may influence client's current or future actions (see Section 1.2.6). It is the second step in meeting the goals of the generalized concept: Management of Present and Future Concepts. The final behavior, Response/Adaptation, provides a policy-based procedure to semi-automate responses to changes that may influence their current or future actions, which is also a basic objective of the EDRA framework (see Section 1.2.6). In terms of core functionality provided by an EDRA solution to clients, the last two behaviors provide for the lion's share. However, the first four behaviors provide the essential ground work necessary for defining the last two.

The rest of this chapter will discuss these six behaviors in detail with references to EDRA's structural components and their interfaces from the previous chapter. John's hospital scenario will be developed into a complete case study through the examples generated to demonstrate EDRA's behaviors in action. The behaviors listed are the minimal set for achieving the goals set out for the EDRA framework. Additional behaviors may be incorporated into the framework for deployment of an EDRA solution in a certain application domain. Even the behaviors listed here could be modified provided the changes are consistent with the complete behavioral design.

4.3 Access

Clients will first require the ability to access the EDRA Core. In Chapter 3 we stated that clients will 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 can 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.

Since we have assumed that a deployed EDRA solution will be delivered as a hosted solution it will publish its connection point(s). For example, EDRA could be hosted using a web server and the connection point would be a URL. The Data Capture Portal of the EDRA Core handles all accesses. Clients interact with the EDRA Core with the help of a Transcoding Manager (TManager) or a Compatibility Manager (an application adapter).

The Data Capture Portal continuously listens for new connections from access devices or application adapters. Based on the connection request, the Data Capture Portal will determine whether it was from an access device or application adapter. One way we can distinguish between a connection request from an access device and an application adapter is by using different connection points. This same technique can be used to distinguish among the various access devices the client may use. For example, if a client uses a cell phone they might find it appropriate to dial a number to access EDRA whereas if they use a PC they can connect via the internet using a URL. Other possible ways of distinguishing among access devices would be connection properties such as the signal strength, protocols, message paths, etc., or even the access devices providing identification information when making the connection request. Figure 4.2 provides a visualization of entities involved when clients access EDRA.

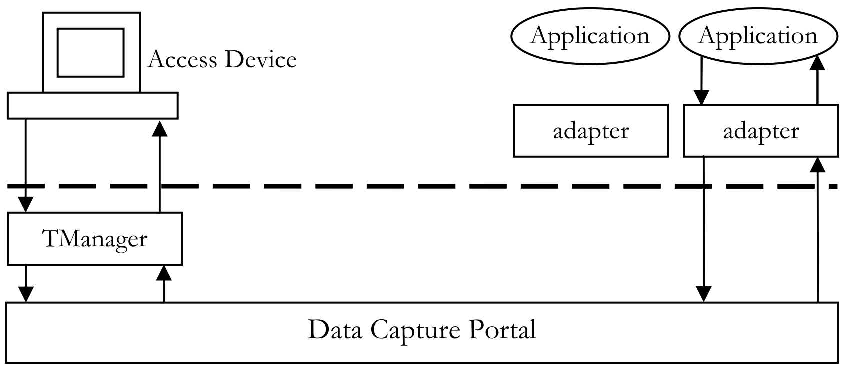


Figure 4.2 Accessing EDRA

If an access device was the initiator of the interaction, the Data Capture Portal will instantiate a TManager to handle the interaction. The Data Capture Portal searches a pre-loaded list of transcoding algorithms to find the appropriate one for the access device being used. The TManager is customized during instantiation with the transcoding algorithm selected by the Data Capture Portal. If, however, an application adapter was the initiator, then access is directed to the EDRA service instance identified in the information piggybacked along with the connection request.

The Data Capture Portal keeps track of all the open connections. A TManager or an application adapter must signal the Data Capture Portal to end a connection on behalf of the client. Upon terminating a connection from an access device, the Data Capture Portal removes the corresponding TManager.

It is now our intention to illustrate how this behavior allows for a non-intrusive, simplified information gathering method. In our hospital scenario, as soon as John entered the hospital a nurse makes an access to EDRA to register a new EDRA service instance for him. Nancy, the healthcare specialist, can then access it on John's behalf to initially add the treatment schedule or later to update it. It is important to observe that registering a patient or creating a treatment schedule are required actions in present day hospitals. Incorporating an EDRA solution doesn't require any adjustments on the part of the actors. When the nurse is registering John an access device such as a PC could have been used with a wired connection to EDRA's hosted solution (in the future possibly a PDA with a wireless connection). A TManager, instantiated with a PC transcoding algorithm would have handled the interaction. Once the nurse has created a registration file the connection is closed. Subsequently, after Nancy checks the extent of John's injury, she may use a specific application to generate the treatment schedule. Providing information about John's EDRA service instance, that treatment schedule is transferred to EDRA Core. The application uses its EDRA adapter to create a connection, transfer the data and then close the connection.

Access only specifies the creation and termination of the connection upon initiation from a client. Other behaviors of EDRA define the details of how interactions take place.

4.4 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 (partial support for Client Control of Client Context concept).

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. Referring back to John's visit to the hospital, an EDRA service instance is created for him upon entry. The nurse responsible for recording his personal information (e.g. medical history, family doctor, health insurance number, etc. that pertain to the healthcare industry in addition to the minimal set of information required) will enter it into EDRA. As will be mentioned in the next behavior EDRA provides the ability to export this information to other applications, such as bookkeeping.

At registration time the owner for the new EDRA service instance must also be provided. The owner may or may not be the client. For example, in our hospital scenario the owner of John's EDRA service instance would be Nancy. Nancy will make all access to John's EDRA service instance on his behalf and may share access with other healthcare specialists involved in his treatment.

Registration for a new EDRA service instance is currently only defined when a client makes an access via an access device. This implies that registration cannot be done through an application. 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. Before being aware of those applications EDRA will not be configured to listen to connections requested by them. Clients will need to configure the application adapters associated with these applications with information about the client's service instance for direct connection and authentication.

Once the initial access by the client or on behalf of the client is made a TManager instance is created as defined by the Access behavior. Recall that in Chapter 3, we stated that the Transcoding Manager can handle information flows from the client to the EDRA Core as well as to the client from the EDRA Core. The TManager's transcoding algorithm uses a form approach to acquiring information from the client. Based on the information required (specific to application domain and a deployment-time decision) a form is generated with the corresponding fields and presented to the user. Clients must enter information that corresponds to fields in the form.

In Chapter 3 we stated that the TManager must be cognizant of the procedures the Data Capture Portal of the EDRA Core. The TManager upon starting a session presents a static Start form which allows the user to either begin the Registration procedure or Log In procedure. The

Registration procedure instantiates a new Response Platform and Context Container and associates them together to create a service instance. Information to be stored in the Client Record, a subcomponent of the Context Container in the newly created EDRA service instance, is required. The TManager presents the information form to the client. Since the data entered by the user is structured in the form, it can easily be entered into the Context Container that was just initialized. The TManager's Registration form will also provide clients the option of providing references to application adapters they want to register with their service instance. If clients exercise this option, the TManager invokes the Application Registration procedure. Once again, a form is generated based on the type of information required about the adapter so that EDRA can recognize it. The information needed to generate the form has to be established at deployment-time. The Data Capture Portal maintains knowledge of the various types of adapters and the information it needs about them to accept connections. Figure 4.3 depicts the form-based Registration behavior. For example, connection information (e.g. port number, location, etc.) needed to invoke the adapter and the application it serves via EDRA. That completes the Registration procedure. The TManager ends the access session or redirects the client to the Start form.

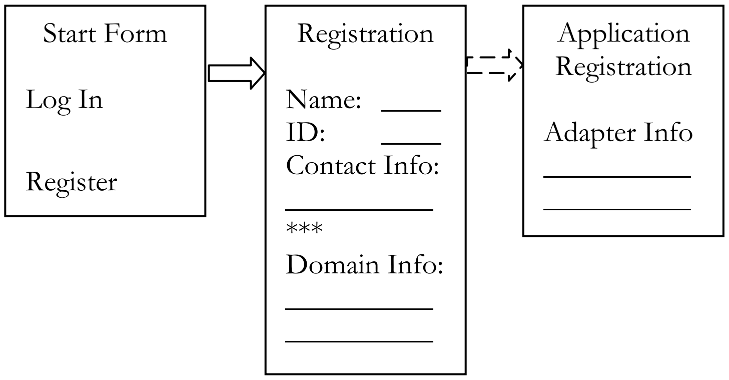


Figure 4.3 Form-based Registration

4.5 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 or owner (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. 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. If the client is not the owner of the EDRA service instance then the owner is required to determine the current and future contexts of the client. In the hospital scenario, the healthcare specialist, Nancy, is the only one who has the knowledge of what John’s treatment schedule should be. As she creates it, she is laying out his contexts while in the hospital.

An EDRA service instance can acquire the preliminary set of contexts for the client either from specialized applications or manual entry by the client/owner using an access device. Specialized applications can include travel booking applications or a hospital treatment specification application that transfer information to the EDRA service instance. The applications communicate with the EDRA service instance through Compatibility Manager Adapters. The adapters manage the communication and guarantee that information flowing into an EDRA service instance is compatible with the corresponding EDRA Data Model for the application domain. It is the adapters’ main function to transform data from applications into Entries or vocabulary specified in the IVTS of the application domain (see Section 3.2.2). The transformation of data by the adapter is beyond the scope of this thesis since it would vary with the application being served by the adapter. Clients/owners that choose to enter the contexts manually using an access device will have their interaction managed by a TManager. Similar to the Registration behavior, a form-based approach guides the entry of information necessary for entering contexts into the EDRA service instance. The TManager generates forms dynamically using information from the IVTS for the application domain of interest for the client. The initial form generated by the TManager makes use of the IVTS’s operations interface to allow the user to search for Entries that match their current and future contexts. Ensuing forms dynamically generated by the TManager will request the user to provide information for the fields of the Entries selected. Once an EDRA service instance receives one or more contexts in an EDRA Data Model compatible format they are stored in the Agenda subcomponent of the Context Container associated with that EDRA service instance.

Assuming a specialized application is used by the client/owner, the application’s adapter makes an access to the Data Capture Portal of the EDRA Core. The Data Capture Portal finds the EDRA service instance the access is meant for. Recall that when making a connection request (Access behavior) the adapter piggybacks client authentication information. The adapter then exports information to that EDRA service instance. The Data Capture Portal imports the information about the contexts in the form of Entries and inserts them into the Agenda subcomponent of the Context Container associated with the EDRA service instance. This is possible only because the information received from the adapter corresponds to the IVTS for the application domain. The adapter can then export information about dependencies among the Entries. The Data Capture Portal imports this information and again inserts this information into the Agenda subcomponent (the Dependency List) of the Context Container.

The Data Capture Portal provides an *Export* operation to provide external application and services information stored in an EDRA service instance’s Context Container. Adapters for specific applications may or may not support it depending upon what their purpose is and if they require information about the client’s EDRA service instance to fulfill that purpose. A good example would be the bookkeeping application in the hospital that was mentioned in the previous section. It would require information pertaining to the patient and the treatment they receive. When requesting information from EDRA on behalf of an application, an adapter would provide an indication of the data it wants (e.g. Entries, Dependencies, Client Record, etc.). The Data Capture Portal would access this information using the Context Container's interface and export it to the adapter. It is the adapter's job to transform the data into a suitable format for the application it serves.

Using an access device to enter contexts into an EDRA Service instance is possible and would begin with an access to the Data Capture Portal. The TManager instantiated to handle the interaction will present the user with the Start form as discussed in the Registration behavior. This time the user will need to select the Log In procedure. The client/owner must identify the EDRA service instance to be accessed. After authentication the current contents of the Agenda subcomponent of the Context Container associated with the EDRA service instance being accessed are presented. The client/owner has the option of adding new Entries. The Add Entries procedure of the Data Capture Portal is followed by the TManager. During this procedure, forms are dynamically generated as explained earlier to guide the client/owner's input to correspond to the EDRA Data Model. Once the client/owner has entered all the required information a newly created Entry is added to the Agenda subcomponent of the Context Container of the associated EDRA service instance. This process can be repeated for however many contexts the client/owner wishes to add. Dependencies among Entries can be entered as part of the Add Entries Procedure or separately using the Add Dependencies procedure. For identifying dependency the client just has to identify two Entries (one is dependent on another). This process can be repeated. Once the client/owner is finished the access device can terminate the access.

Examples to illustrate the two ways an EDRA service instance can acquire the client's current and future contexts can be provided by the hospital scenario. Nancy can use the hospital's treatment specification application to create the treatment schedule. The application will transfer information to its Compatibility Manager Adapter, which in turn will transfer that information (after converting it to an EDRA Data Model compatible format which in this case would be based on HL7 [12]) to the Data Capture Portal of the EDRA Core. The Data Capture Portal will then enter the treatment schedule into the Context Container associated with John's EDRA service instance. If Nancy forgot to add one or two treatment contexts she can directly access John's EDRA service instance using a PDA. A TManager instantiated with a PDA transcoding algorithm will handle the access. She will then add new entries one by one and possibly indicate dependencies. When finished she will end her interaction.

So far we have only considered the fact that the current and future contexts are set by the client or owner of the EDRA service instance as a preliminary step. However, the clients or owners may need to add new contexts while their EDRA service instances are in operation. For adding more contexts, the same process mentioned previously is used. Clients and owners also may wish to update or remove certain contexts from their EDRA service instances. For these interactions the process is similar except that the Add Entries procedure is substituted with the Update Entry (modification of information for the Entry's fields) or Delete Entry (removing an Entry from the Agenda) procedures respectively. For altering dependencies an undesired dependency must be removed using the Remove Dependency procedure and new ones can be incorporated using Add Dependency procedure.

4.6 Initialization

This behavior defines how an EDRA service instance prepares itself for the 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 which is shown in Figure 4.4. This is one of the main contributions of this thesis. Also, the contexts within the EDRA service instance will be partially ordered after the addition of new contexts, updates, or removal. The Service Repositories subcomponents of the Context Container associated with the client's EDRA service instance will be updated during this behavior.

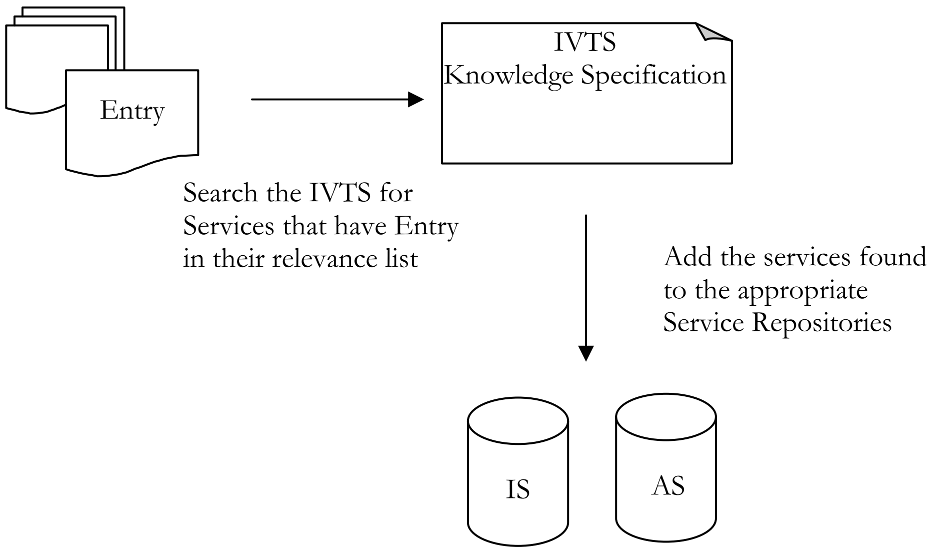


Figure 4.4 Selection of relevant services

Recall that each Context Container is affiliated with a Response Platform to form an EDRA service instance. The Data Capture Portal invokes the *Populate_IS* and *Populate_AS* operations of the Responder component that drives the Response Platform. This is done immediately after the Acquisition-of-Contexts behavior for a specific EDRA service instance. Both operations are supported by the facilities the EDRA Data Model provides. We already documented that the IVTS of the application domain lists the possible Entries and all available services in the application domain (see Section 3.3.2.1). Additionally, the services named in the IVTS's knowledge specification maintain a list of all Entries for which they are relevant. A client's current and future contexts (schedule) are represented as Entries in the Agenda subcomponent of the Context Container associated with the client's EDRA service instance. The Responder can use the IVTS's operations interface to search for all services that are declared relevant to Entries in the client's schedule. The *Populate_IS* and *Populate_AS* operations make use of the *Get_IS* operation and *Get_AS* (see Section 3.3.2.2) operation respectively. The Responder's *Populate_IS* and *Populate_AS* operations then add these relevant services to the affiliated Context Container's Service Repositories. If a service has already been stored in a Service Repository then only a reference of the Entry is added to the service.

The Responder repeats this process for each new Entry that was added, during the Acquisition-of-Contexts behavior.

Contexts may have been updated during the Acquisition-of-Contexts behavior and that means the corresponding Entries were updated. *Populate_IS* and *Populate_AS* initiate a search of all the services in the IS Service Repository and AS Service Repository for references to the Entry that was updated. References to the Entry by services are removed and if a service has no other references to other Entries it is removed from its Service Repository. This is done because previously relevant services may no longer be relevant. *Populate_IS* and *Populate_AS* can then treat the updated Entry as a newly added Entry and follow the process outlined before to determine the relevant services after modification. If contexts are removed during the Acquisition-of-Contexts behavior then *Populate_IS* and *Populate_AS* would remove references to that Entry by services in the respective Service Repositories.

When new contexts are added they are partially ordered in the Agenda subcomponent. At present ordering is achieved through explicit start and end times specified by each Entry representing a context. It is a partial ordering because one or more Entries may have overlapping time durations. Potentially a defined priority scheme and the dependency list can also be used for ordering the Entries. The ordering is managed by the Context Container when Entries are added, removed or when dependencies are changed in the Agenda subcomponent.

Let us now look at this behavior in action within our healthcare scenario. Once John's treatment schedule is entered into his EDRA service instance the IS Service Repository and AS Service Repository in the associated Context Container are populated. The hospital's IVTS defines all the services available and establishes relevance of those services to Entries, which are also, defined in the hospital IVTS. Within EDRA John's treatment schedule would be represented as a set of Entries and the hospital IVTS would be searched for services relevant to them. The Responder of his EDRA service can identify relevant IS services for his first X-Ray room context – X-Ray result service, Operational Status Update service (Used to send status changes of the X-Ray room – In Service, Maintenance Required are possible status values), among others. It will also identify relevant AS services for his X-Ray room context – X-Ray room booking (used when original booking is cancelled due to serving emergency patients) and possibly others. In a large hospital there may be more than one X-Ray room, therefore, Nancy can modify John's X-Ray room context by changing the room number. John's EDRA service instance would then remove references to the Entry by services already in the Service Repositories (e.g. Operational Status Update Service since it is room specific). It would then find the other services that would be relevant for the updated changes (e.g. Operational Status Update Service for the new room).

4.7 Monitoring and Management

Once an EDRA service instance for a client has been initialized, control is switched over to the Monitoring and Management behavior. This behavior is actually realized through three sub-behaviors. The first is the IS Relationship Establishment, which is responsible for establishing dynamic service relationships with relevant IS services. The second is Channeling Events, which directs updates received from IS 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 the transition from one context to another. Together these sub-behaviors provide the means to preemptively identify changes in a client's schedule.

4.7.1 IS Relationship Establishment

In the Initialization behavior the Responder's *Populate_IS* operation was responsible for adding all the relevant IS Services for a given context into the IS Service Repository in the affiliated Context Container. Before an IS service is actually entered into the IS Service Repository the EDRA service instance must successfully establish a service relationship with the IS service. This sub-behavior actually spans the Initialization behavior and the Monitoring and Management behavior.

The Responder's *Populate_IS* operation employs the *Subscribe* operation of the corresponding EMM component in the Response Platform. Recall that the *Subscribe* operation initiates the Subscriber subcomponent of the EMM to set up an Event-Listener to listen for notifications from the IS service. If this is successful then the Responder will add the IS service into the IS Service Repository of the affiliated Context Container. Otherwise, that service is not added into the IS Service Repository. If the responder was unable to initiate a service relationship with an IS service for a future context it keeps track of it until the future context becomes the current context (transitioning from one context to another will be explained in Section 4.7.3). At that point, it will once again attempt to initiate a service relationship with that IS service and if successful will add it to the IS Repository.

It is important to note that all IS services are required to have a common set of operations (e.g. Subscribe, Unsubscribe) which facilitates dynamic service establishment. The IS services may differ in their actual operations signature, which will include information required by the IS service from the subscribing entity. Notifications from the IS services will be in an EDRA Data Model compatible format which can be processed by the Responder. Notifications will be discussed in more detail in the Channeling Events sub-behavior.

Referring back to John's situation, his EDRA service instance might find a heart-beat monitoring service for his room relevant for his first Rest context before surgery (associating relevance between services and contexts is explained in the previous section 4.6). It will then attempt to establish a service relationship with that IS service. However, since John only has hurt his knee and is not in a critical condition, the room where he will Rest will have the heart-beat monitoring service turned off. The heart-beat monitoring service, therefore, will not be added to the IS Service Repository of the Context Container associated with his EDRA service instance. If we assume that John has surgery, then in the second Rest context after surgery the service would be activated by his healthcare specialist. This time the Responder of the Response Platform associated with his EDRA service instance will successfully subscribe to the service and add it to the IS Service Repository of the corresponding Context Container. The Subscriber subcomponent of the EMM associated with his EDRA service instance would create an Event Listener to listen for changes in his heart rate.

4.7.2 Channeling Events

An EDRA service instance is in the monitoring phase when it is waiting for change updates to be received by an Event Listener from subscribed IS services. This sub-behavior specifies how these updates are received and channeled to the appropriate Entries in the Agenda subcomponent of the Context Container associated with the EDRA service instance. We will refer to the change updates received from IS services as events. The response to events will be part of the Response/Adaptation behavior since it is exception handling rather than monitoring and management.

Each Event-Listener is mapped to a specific IS service. All the Event Listeners wait to receive events from the IS services they are listening to. Once an event is received it is in an EDRA Data Model compatible format. For example, the event could consist of an Entry with specific values for properties of that entry. Since the Entry was received as an event the values sent must reflect a change. Entries represent the operating environments of a context so it would be a reasonable approach to constructing events. It is required that IS services will communicate with EDRA in an EDRA Data Model compatible format.

An Event-Listener upon receiving an event would place the event in the Buffer component of the Response Platform associated with the EDRA service instance. The Responder then retrieves the event from the Buffer to process the event. It is known which IS service transmitted the event since there is a one-to-one mapping between a subscribed IS service and an Event Listener. Using this information the Responder knows which Entries in the Agenda subcomponent of its affiliated Context Container, need this update.

In the healthcare case, let us observe this behavior in action for John's Surgery context. The IS service of interest is the Availability-of-Theater service. This would provide updates when there is high demand for the operating theater booked for John. In case of emergencies, John would be bumped from his time slot. If on that day an emergency case arrives and the operating theater becomes booked the Availability-of-Theater service would send an event to John's EDRA service instance. An Event Listener would receive it and place it in the Buffer. The Responder would retrieve the event from the buffer and deduce that this event is directed to John's Surgery context.

4.7.3 Transition

An EDRA service instance moves into the management phase when the client makes a transition from one context to another. Within EDRA this is depicted as a transition from one Entry to another in the Agenda subcomponent of the Context Container associated with the EDRA service instance. The Transition sub-behavior defines how this takes place. Transition only defines behavior when there are no changes influencing the prescribed schedule. Transition occurs not due to a received event from an IS service, but because time elapses. The Response/Adaptation behavior handles deviation from the prescribed schedule due to the receipt of a change event.

The Response Platform of an EDRA service instance by default keeps track of time. When an Entry in the Agenda subcomponent of the affiliated Context Container reaches its end time the *Respond* operation of the Responder undertakes the Transition behavior. There are two possibilities.

The first possibility is that Entry that just expired is an operating environment for a context that will not be repeated. The second possibility is that it will be repeated. The Responder will first remove references to IS and AS services in the Service Repositories for that Entry. If the Entry is not repeated again it is purged from the Agenda subcomponent of the affiliated Context Container. If, however, the Entry is to be repeated (e.g. a cyclical schedule) then it would be added into the Agenda as a new Entry with the appropriate new start and end times, and possibly revised values for other contents of the Entry. An Entry will need to be repeated if it still has dependencies stored in the Agenda's Dependency List. Entries that make up a cyclic schedule must provide start and end times allowing the Responder to properly revise the contents of the Entry (e.g. An Entry can be given a start time equal to the end time of the last Entry in the cycle and its end time can be calculated using the duration).

Transition sub-behavior can potentially be redefined for addressing transitions of context not explicitly based on time. Also, the Transition sub-behavior here just attempts to mimic real world context changes and does not induce the client to change contexts. There are certain instances when a priority scheme for transitioning among contexts is more reasonable than strict transitioning based on start and end times. For example, in a travel scenario, a client must commute to a place by a certain time and the estimated duration for the commute can be retrieved. The priority of the commute will increase as the time gets closer until user has to enter the commute context in order to get to the destination. This type of transitioning does not necessarily deal with the removal of preceding contexts but focuses more on informing the client on changing the current context based on calculated priorities. The client's preceding context may be postponed or removed. This is an example of how the behaviors discussed in this chapter may be refined, or even redefined to meet the needs of the application domain.

Revisiting the healthcare case, we see that transitions take place between each of John's contexts. When John completes his surgery, he will need to transition to his Rest and medicine context. Within his EDRA service instance, the Surgery Entry is removed from the Agenda subcomponent of the Context Container by the Responder component of the Response Platform.

4.8 Response/Adaptation

This behavior is geared towards defining how an EDRA service instance can semi-automate a response and help clients adapt to changes that occur in the operating environments of their current and future contexts. This entire behavior is made possible by the existence of Contingency Policies. Recall from the previous chapter that there are three levels of contingency policies – Industry-level defined in the IVTS (EDRA Data Model) for an application domain, Agenda-level, and Entry-level. This behavior follows right after the Channeling sub-behavior of the Monitoring and Management behavior. Response/Adaptation has to balance simplification of processes through automation while also allowing for the client to still maintain control over changes to their current and future context. This is the reasoning behind three-level, policy-based response construction and the semi-automation of the adaptation procedure (as supposed to complete automation).

To semi-automate a response to a given event, the Responder component invokes its *Respond* operation. This first searches the contingency policies of the Entry for which the event is directed (Finding the Entry for which the event notification is relevant was deciphered during the Channeling sub-behavior). The name of an Entry's property (a content field of the Entry tuple, see Section 3.3.2.1.1) that changed and its new value are used as the search key. If there is no contingency policy at the Entry-level to deal with a particular event and the change reflected by the event, then the Respond operation searches the Agenda-level contingency policies. Bear in mind that the client can specify contingency policies and/or import them at the Agenda-level. Again, if no contingency policy exists for the event and the change it reflects at the Agenda-level, the Industry-level contingency policies are searched.

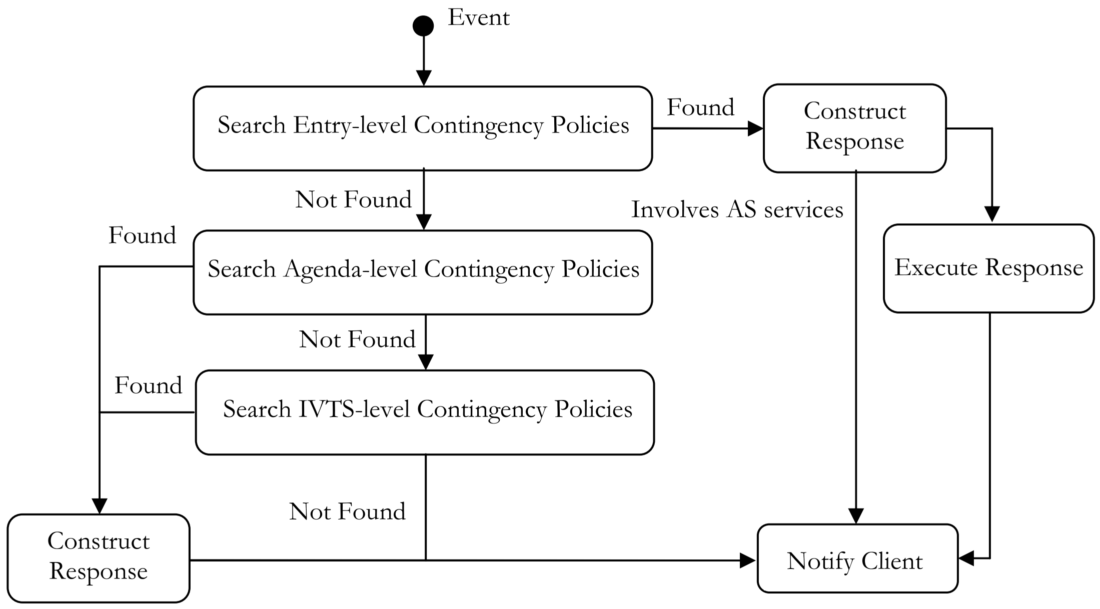


Figure 4.5 Response/Adaptation decision making

A response is constructed in the following manner and the decision making process is depicted in Figure 4.5. If an Entry-level contingency policy is found for an event and for the change it reflects then the response is specified by that policy. The catch-all default contingency policy for every Entry is to notify the client. As a result, if only an Agenda-level contingency policy that was imported or an Industry-level contingency policy exists to deal with the event and the change it reflects, then the response is a notification to the client along with all the suggested responses. The suggested responses would be all the policies that were matched. This is because clients may not be aware of the changes that pre-defined policies will make on their EDRA service instances.

Now that the response has been constructed the Responder must semi-automate the client's adaptation to the event. Adaptation can involve modifying the Agenda subcomponent of the Context Container affiliated with the Response Platform the Responder is part of. Modification can take the form of updating the Entries and the Dependency List of the Agenda. The Responder can also modify the contents of the Service Repositories of the Context Container. Responses that only specify modifications of the Agenda are completely automated and are similar to the Responder's actions during the Transition behavior. The Responder could update Entries or the Dependency List in the Agenda subcomponent of the Context Container it is affiliated with. Unlike in the Transition behavior, the client is notified of the updates made by the Responder. The *Notify* operation of the EMM component associated with the Responder for the EDRA service instance is used. For example, when John is being monitored and given medication after his surgery the Patient Recovery Monitoring Service would send an event notification when his recovery rate changes (John is healing faster than expected). This could require an update of the current context to decrease the dosage of the medication to be supplied. Given that Nancy had specified this contingency policy it would not require Nancy's intervention to take effect. Nancy will still be notified of the changes that were enacted.

A constructed response may also have to make use of relevant AS services for one or more contexts. This is required when Entries have to be added or removed from the Agenda subcomponent of the Context Container associated with the client's EDRA service instance. When modification involves the use of AS services, EDRA forces the client to make conscious decisions. In this situation, an EDRA service instance currently only specifies support for the semi-automation of the response. The client is sent the event and the response to be taken by the *Notify* operation of the EMM component associated with the Responder of the EDRA service instance. The client is required to access their EDRA service instance, and view the most recent suggested response. If in agreement the client can choose to execute the response. The Data Capture Portal will then guide the client through the response by invoking AS services. Note that the client makes the access via a TManager, and the Data Capture Portal invokes the AS service using Compatibility Manager Adapters. The Data Capture Portal must route client inputs from the TManager to the AS service adapter, and outputs from the AS service adapter to the TManager for the client. If a client's EDRA service instance can only provide suggestions for a response then a similar process is followed, except the client has to choose which response to execute, or construct a new response. If the client chooses to construct a new response, the relevant AS services are available to do so. The client can also choose to invoke AS services indirectly using the Data Capture Portal as described or invoke them directly and have these AS services export the results to the client's EDRA service instance.

An illustrative example of semi-automating a response would be if the first X-Ray context for John identified no major injury to his knee. Nancy may have provided a contingency policy to skip the Preparation-for-Surgery and Surgery contexts. Since the Preparation-for-Surgery and Surgery contexts require reservations, Nancy must un-reserve her prior request. John's EDRA service instance notifies Nancy and provides her the response to take. She can invoke the hospital's various reservation services via EDRA to make her changes. The modifications of the Agenda subcomponent of the Context Container associated with John's EDRA service instance are automated.

In Chapter 1, we stated that EDRA allows for transient service relationships. As contexts are purged from the Agenda subcomponent (Transition and Respose/Adaptation behaviors) all relevant IS and AS services are also removed from the Service Repositories, unless other contexts also consider them relevant. When an IS service is removed the EDRA service instance's Responder instructs the Subscriber subcomponent of the EMM to use the *Unsubscribe* operation to end the service relationship and terminate the Event Listener mapped to the IS service.

4.9 Chapter Synopsis

This chapter provided an in-depth look at the behavioral design of EDRA. We began by introducing a case study to provide a concrete basis of understanding for the reader. Then the six behaviors that govern EDRA's operations were discussed. Our discussion of the behavioral design included tracing the flow of information, and interactions among the structural components of EDRA. The case study was revisited for each behavior to showcase how EDRA would function in a possible deployment scenario. The next chapter will analyze the structural design presented in Chapter 3 and the behavioral design presented in this chapter.

Chapter 5

Design Review

After spending two chapters detailing the structural and behavioral design of the EDRA framework we can now take a step back and review it as a whole. The discussion will focus on the modularity, flexibility and maintainability of EDRA. This analysis will be helpful in corroborating our assertions that EDRA meets the objectives established in Chapter 1. We also cover the features of EDRA that enable it support incremental adoption.

5.1 Modularity

We define modularity as the degree to which a functionality contributed to the overall system by specific component is self-contained within that component. A component in a modular system has distinct functionalities and is not dependent upon other components to provide them. Modular components can compose their functionalities to provide a higher level functionality. The manner in which components compose their functionalities can also influence modularity. Composition should occur through clear well defined interfaces. Modularity allows changes to components of the system to be localized. This eases the incorporation of enhancements to functionalities of the system with minimal reconfiguration.

A close observation of the EDRA framework would reveal that modularity was a corner stone of the design. EDRA Periphery provides client interfacing functionality, EDRA Data Model encapsulates the template of standardized data in an application domain and the EDRA Core provides for the storage of clients contexts, monitoring of changes and semi-automation of responses to changes. Modularity was emphasized even within these main classes of design elements. The EDRA Periphery divided its responsibility among the Transcoding Manager and the Compatibility Manager, each dealing with a separate types of client initiated interaction. The EDRA Data Model kept the specification of the standardized data distinct from the operations on the specification. Similarly, EDRA Core delegated its functionalities to the Data Capture Portal, Context Container and the Response Platform. The Data Capture Portal handled access to EDRA service instances, Context Container provided storage facilities and the Response Platform took responsibility for monitoring and semi-automating responses. If we wish to dwell deeper we will find that even the elements that make up the Context Container and the Response Platform were also designed in a modular fashion.

If we were to improve the modularity of the EDRA framework we would first concentrate on the Data Capture Portal. At present it meets its requirements as the point of entry into the EDRA Core. However, there is room for improving how it provides its functionalities. We feel that it could be better engineered internally by segregating among the distinct processes it is tasked to handle (e.g. client interaction, interaction with EDRA service instances). This would be best done during implementation since all the processes to be managed by the Data Capture Portal will be explicitly detailed.

The intent for designing the EDRA framework in a modular fashion was not just for following software engineering guidelines. We were constructing a generic framework for various application domains which should be capable of handling a multitude of scenarios within these application domains. Only a modular design would allow for swift customization.

5.2 Flexibility

For our purposes flexibility is defined as the extent to which the modularity of a system is utilized to support the necessary diversity in operations. Where modularity focuses on the functionalities supplied by components in the system, flexibility is geared towards assessing the variety a functionality can deliver. Flexibility also takes into account the adaptability of the system. This trait is a necessity when proposing a generic infrastructure because it needs to be customizable for specific needs.

We can endorse the EDRA framework's ability to capitalize on its modular design to achieve the necessary level of flexibility by highlighting its capabilities. Modularity does not guarantee flexibility. It is necessary to identify the functionalities and associate them appropriately with design elements so that a modular system can provide for the types of flexibility required of the system.

An EDRA solution has to be accessible via a multitude of access devices and applications. The EDRA Periphery makes this possible. The Transcoding Manager can support new access devices by incorporating the appropriate transcoding algorithm. Selection of the transcoding algorithm corresponding to the access device used will occur at runtime dynamically. Applications can also interface on behalf of clients with EDRA by making use of an adapter. Given that the framework has no control over the type of devices or the applications that clients will intend to use with EDRA, this is a flexible manner by which they can be supported.

We can observe some of the adaptability features of the EDRA framework. Adapting the EDRA framework for different application domains is achieved by employing the IVTS corresponding to that application domain as the EDRA Data Model. The Context Container of the EDRA Core assists in supporting any scenario that can occur within an application domain. The Agenda subcomponent can store any set of Entries that are defined in the IVTS of the application domain in any partial order to model the scenario of the client. The Context Container and the Response Platform of the EDRA Core together allow for dynamically adapting to changes that may occur in the client's Agenda. The methodology followed by the Response Platform and the contingency policies maintained by the Context Container make this possible. Improving upon the adaptability of the EDRA framework would first focus on adding support to more than just time-based Entries for modeling a client's scenario.

The EDRA framework's flexibility is also apparent in deployment tradeoffs. It was mentioned in Chapter 3 that the EDRA Data Model has two deployment options. In the first the EDRA Data Model's operations interface can be hosted with the EDRA Core and in the second it can be deployed as a service. The benefit of the first is that all interpretation of data can be done by the operations interface and the same operations interface can be used with different knowledge specifications. The advantage of the second option is that other services besides the EDRA Core may be able to use the facilities provided by the operations interface. A third deployment option may also be possible. The EDRA Data Model can also be deployed in a hybrid configuration where a default operations interface can be deployed as a public service and specialized operations interfaces can be hosted with the EDRA Core where required.

5.3 Maintainability

We consider maintainability to be the measure of how well the modularity of the system will enable it to evolve over time with changing technologies and requirements. Flexibility takes into account functionalities provided by the system at one point in time but maintainability is concerned about the ability of the system to improve the functionalities and add new ones. Maintainability can only be an approximate measure taken within the constraints of how we believe the system will be used for in the future and what trends we anticipate will succeed.

Over a short-term we can expect that the number of new services will increase. The providers of these services will want exposure to clients. In EDRA new IS and AS services can be added into the EDRA Data Model after they are certified. The service providers only need to state the categories of the services they provide and what Entries these services are relevant for. The next time a client adds an Entry into their EDRA service instance any newly added services that are relevant are brought to their attention. Only the knowledge specification of the EDRA Data Model is modified. For longer running schedules it may be necessary to make Entries in the client’s Agenda aware of newly available services. This can be achieved by requiring the Response Platform to maintain an Event Listener to changes in the EDRA Data Model’s knowledge specification. The Agenda subcomponent of the Context Container associated with the client’s EDRA service instance would specify a default contingency policy. The policy would require an update of the client’s Service Repositories to include new services if they are relevant to any of the Entries already stored in the client’s Agenda.

When deploying EDRA in an open environment like the travel industry the incentive for providing services would be greater if the service providers are compensated for their effort. At present the compensation would be in the form of greater customer satisfaction (ex. Airlines – flight information, Hotels – hotel booking etc.) which leads to customer retention and service differentiation. But subsequently to increase the variety of services (ex. Third party providers) and achieve higher levels of Quality of Service, we might need to support monetary compensation (ex. micropayments). The Response Platform’s Subscriber subcomponent can be tasked with negotiating price/performance with service providers. Services published in the EDRA Data Model can also document the QoS levels they provide in addition to other details about the service. The Context Container can maintain information about clients that can be used by service providers to bill them. In addition to maintaining contingency policies, the Agenda subcomponent of the Context Container can maintain policies regarding QoS specified by the client.

When considering much longer-term changes we must anticipate technology trends that may have to be supported by the EDRA solution. The growing of interest in the semantic web and its promises for enhancing web-based distributed computing is one such trend. The EDRA framework can be evolved to take advantage of the capabilities of the semantic web. If an EDRA solution was deployed today, we would assume that the EDRA Data Model would use XML schema language for documenting the knowledge specification. In the future, to interoperate with resources in the semantic web, the EDRA Data model's knowledge specification would maintain a set of ontologies that describe the Entries in the application domain of interest. IS and AS services in the knowledge specification would have semantic annotations that would relate them to Entries they are relevant to. Also, we can use a formal XML based calculus to define event-condition-action contingency policies and use logical reasoning in the Response Platform to make response decisions. Due to the design of the EDRA framework the bulk of the modifications are localized to the EDRA Data Model and Response Platform.

5.4 Incremental adoption of an EDRA Solution

In this section we substantiate our claim that EDRA was designed for incremental adoption. When designing EDRA we began by distinguishing the traits that inhibit adoption of a distributed solution. The next step was to examine why these traits were a hindrance. The following is a list of our findings:

1. High initial overhead: Potential end users must not incur large set-up costs (ex. time, efforts) which can diminish the appeal of productivity solutions.
2. Technology dependency: A solution tied to specific technologies has numerous deficiencies. A client needs to be shielded from technology requirements. The solution itself should not require more then the current mainstream proven technologies for initial deployment and be able to incorporate new technologies. If the solution is tied to a technology it requires client adoption of the technology before adoption of the solution and leads to high initial overhead. Its lifetime is tied to the lifetime of the technology and may stifle adoption.
3. Peer-dependent adoption: Minimal dependency on the overall level of adoption and auxiliary support to make use of the solution. On one level it means that in a closed environment (ex. a company, an individual) can deploy the solution for meeting their needs without external assistance (ex. vendors). On another level it means that in an open environment (ex. across organizations or in an industry) a client's acceptance of the solution should not be dependent on other clients also adopting the solution. If dependency is high among peers for adoption of the solution then there is a cyclical effect that discourages adoption by every peer.
4. Grandiose Deployment: Graduated increases in Quality of Experience guarantees instead of over promising initially will keep help meet client expectations. For increasing the Quality of Experience in a distributed solution requires increasing the number of providers and will only happen over a period of time. Initially clients will experiment with the solution to determine its potential and this will induce more providers to market their services increasing the Quality of Experience. The solution should only guarantee only what is supported at a certain point in time otherwise clients will be discouraged from

using the solution. The risk associated with requiring a large deployment initially will discourage prototyping experimentation which in turn hinders adoption.

To best understand how EDRA avoids these pitfalls we will identify the design decisions made specifically to avoid them.

1. EDRA is to be deployed as a hosted solution. Clients can use their existing access devices to access their EDRA service instance from any location. Only if they wish to use certain applications to feed information into their EDRA service instance, they would need to install an adapter but this too is just a one time affair. An EDRA service instance will also notify clients using their preferred mode of contact.
2. The only technology requirement that EDRA has is that it has to be implemented using SOT. However, SOT is currently the major trend in integration and coordination in heterogeneous environments and remains transparent to the clients. Additionally, EDRA was designed to be SOA agnostic. The software infrastructure that EDRA proposes can be extended to make use of possible new technology initiatives in distributed computing (see previous section – Maintainability).
3. In the EDRA framework a single client's use of the solution is independent of other clients using the solution. EDRA also supports deployment in closed environments as described in the healthcare case study in the previous chapter.
4. To deploy any distributed solution there is a certain threshold that must be achieved to support meaningful activities of a client. In the case of EDRA the providers of the information and action services and the availability of adapters for applications will determine the Quality of Experience. As new providers abound EDRA can improve its Quality of Experience. Standardization of data in an application domain can be the most expensive in terms of time and effort when deploying an EDRA solution. However, the EDRA Data Model can be incrementally upgraded as the EDRA solution is used in different contexts within the application domain reducing the initial investment of time and effort. This encourages prototyping and experimentation to recognize the value of the EDRA solution with minimal resources and if satisfied can be expanded.

5.5 Chapter Synopsis

This chapter reviewed the EDRA framework with attention directed at how its modular design provides for both flexibility and maintainability. A discussion on how EDRA facilitates incremental adoption was also provided. After having delved deep into the details of the EDRA framework it was necessary to take a higher-level perspective to recognize what those details help us achieve.

Chapter 6

Conclusions & Future Work

Our research has resulted in the Event-Driven Response Architecture for Service-based Computing. The trend towards service-based computing has been mainly driven by the need to achieve loose-coupling among entities. We have proposed this software infrastructure as a means to realize the true power of loose-coupling. EDRA allows service consumers to have the facility to automate the selection and invocation of relevant services during runtime. This frees the client from having to be cognizant of all the available services pertinent to their tasks during development time. Even if an SOA allows clients to select and invoke services dynamically, EDRA would free them from having to explicitly provide highly-detailed information to search and invoke services each time they want to employ one. As new services come into existence EDRA can dynamically make them available for use if so required. EDRA recognizes that clients' operations may have to adapt to change over a period of time. Therefore, it automates monitoring of the client's current and future contexts and is capable of semi-automating adaptation responses.

We constructed EDRA to be a generic solution. EDRA can be customized to serve various application domains. An EDRA solution for an application domain is capable of handling diverse scenarios within that domain. Clients only specify the logic of the operations they want to undertake (i.e. the scenario). Over the course of a client's scenario, EDRA manages all service interactions required to monitor for changes and respond.

Research on the EDRA framework has put us closer to fulfilling the concepts introduced in Chapter 1. In contrast to the order in which they were presented, EDRA's behavioral design was actually finalized before the structural design. This was because of the fact that, in order to address the concepts it was essential to first meet the basic objectives (see Section 1.2.6). Reaching these objectives meant outlining the necessary behaviors and subsequently associating the behaviors with required structures. Our software engineering process had significant parallels to the Tropos software design methodology [5]. Though the software engineering exercise was a by-product of having to build a software infrastructure, it was critical for making research contributions. For example, it helped pinpoint the minimum level of standardization required for loosely-coupled interactions. This discovery in turn allowed us to propose the data model for specifying knowledge in an application domain. The data model also served as the infrastructure necessary to correlate client contexts with their relevant services.

6.1 Future Work

Having designed the EDRA framework to meet the objectives and address the research issues outlined in Chapter 1 we now look at enhancing and extending our work. We strongly believe that the effort we invested into the design will encourage future work on the EDRA framework due to its simplicity and clarity. We have already alluded to specific aspects of the EDRA framework that should be revisited. However, here we present the major initiatives that we believe should be the starting point for future work.

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 EDRA Data Models. 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 and also monitor their home environment. If two different EDRA Data Models 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.

The next 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 order 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 process or workflow monitoring.

At present the EDRA framework only semi-automates the invocation of AS services for balancing increased efficiency with control over actions. There are some application domains where complete automation may be more appreciated. As a possible extension to our work, we want to investigate some of our ideas on how to support complete automation of a client response (as specified by a contingency policy) through the invocation of AS services directly. If EDRA is able to invoke AS services automatically, security would also play a bigger role. Detailed analysis of the security requirements has to be made and addressed.

The initiatives that have been presented are just a subset of all the possible future work that can be pursued on the EDRA framework. It is also important to think about integrating research from other groups into the EDRA framework to maintain its relevance in the ever-changing research landscape. For example, there have been various proposals on how to provide service consumers QoS choices before invoking services [29]. This work could be experimented with for inclusion into the EDRA Data Model.

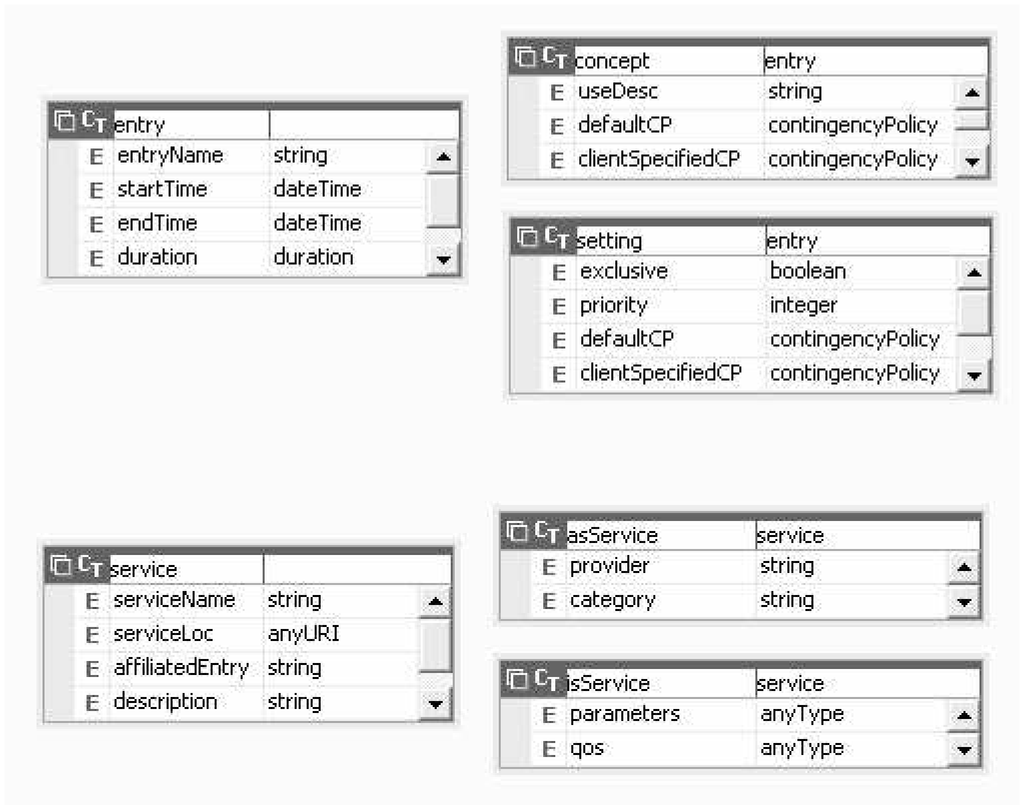
As a closing remark, we recognize that EDRA, being a reactive architecture capable of associating relevance between contexts and services, can be applied in pervasive computing. EDRA can provide clients with awareness of information that pertains to their actions preemptively. The framework is designed for universal access and can semi-automate responses to changes that influence clients' current and future actions. Given that EDRA promotes incremental adoption it can be seamlessly integrated into the existing technology infrastructure for value assessments. The potential for EDRA to provide the software infrastructure required for achieving the pervasive computing vision exists. We now have to consider specific implementations to assess its feasibility.

Appendix A

Sample Implementation of an IVTS Knowledge Specification

The EDRA Data Model as we have discussed is a critical aspect of the EDRA framework. The main challenge is the knowledge specification. Here it is our intention to trace through a possible implementation of the IVTS knowledge specification for the travel application domain. This simple example will provide the reader with a better understanding of the concepts discussed earlier.

In our example we used XML Schema-Definition Language to implement the Travel-IVTS knowledge specification. The root schema as shown in Figure A.1 is the generic IVTS schema. A domain specific IVTS will inherit the basic building blocks of the knowledge specification – Entries and Services – from this root schema. Authors of the domain specific IVTS may redefine or augment these elements as necessary. The figures in this chapter use the following abbreviations for XML Schema components: E – element, A – attribute, CT – complex type, ST – simple type, F – facets.



CT	entry	
E	entryName	string
E	startTime	dateTime
E	endTime	dateTime
E	duration	duration

CT	concept	entry
E	useDesc	string
E	defaultCP	contingencyPolicy
E	clientSpecifiedCP	contingencyPolicy

CT	setting	entry
E	exclusive	boolean
E	priority	integer
E	defaultCP	contingencyPolicy
E	clientSpecifiedCP	contingencyPolicy

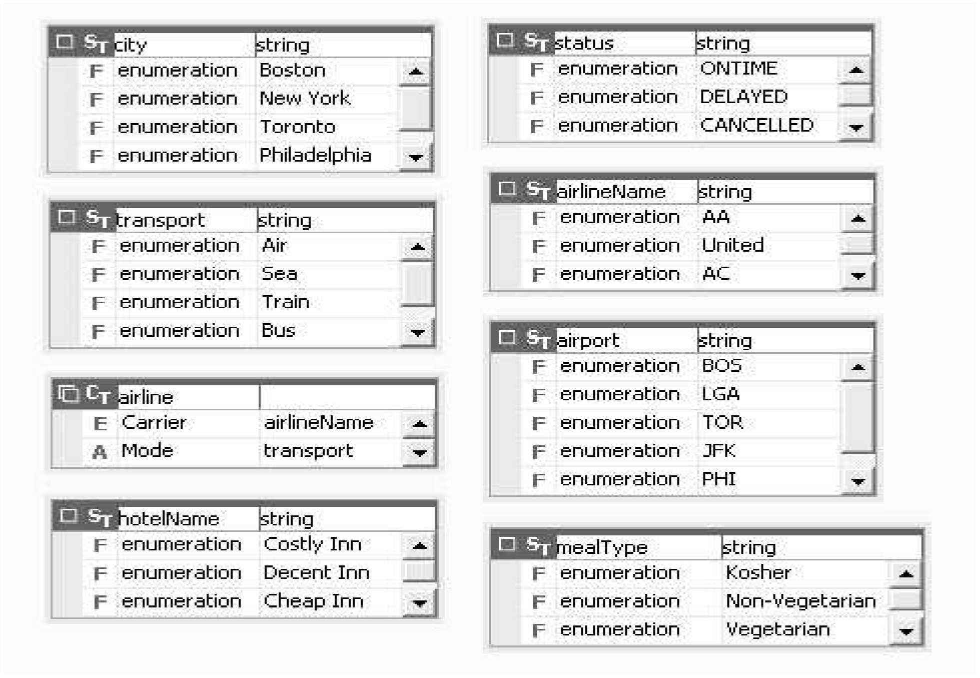
CT	service	
E	serviceName	string
E	serviceLoc	anyURI
E	affiliatedEntry	string
E	description	string

CT	asService	service
E	provider	string
E	category	string

CT	isService	service
E	parameters	anyType
E	qos	anyType

Figure A.1 Root IVTS XSD

In Figure A.1 Entry and Service are roots of an inheritance hierarchy. Also every Service has references to one or more Entries (affiliatedEntry).



ST city	string
F enumeration	Boston
F enumeration	New York
F enumeration	Toronto
F enumeration	Philadelphia

ST status	string
F enumeration	ONTIME
F enumeration	DELAYED
F enumeration	CANCELLED

ST transport	string
F enumeration	Air
F enumeration	Sea
F enumeration	Train
F enumeration	Bus

ST airlineName	string
F enumeration	AA
F enumeration	United
F enumeration	AC

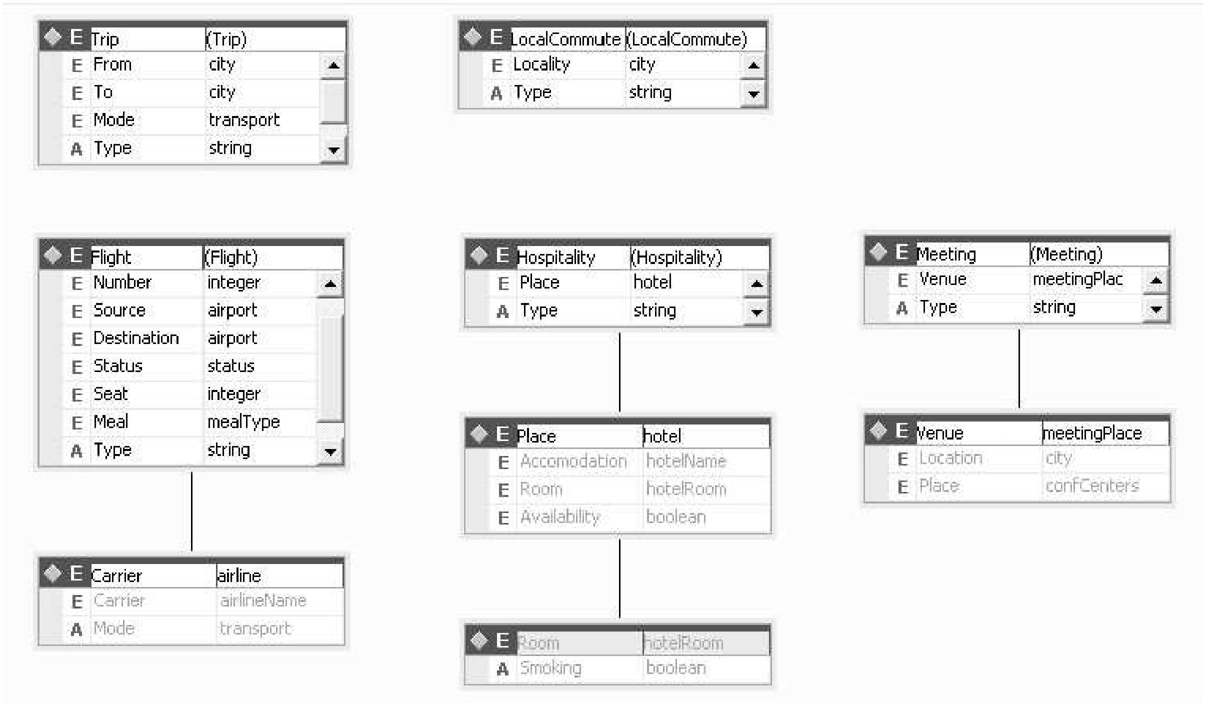
CT airline	
E Carrier	airlineName
A Mode	transport

ST airport	string
F enumeration	BOS
F enumeration	LGA
F enumeration	TOR
F enumeration	JFK
F enumeration	PHI

ST hotelName	string
F enumeration	Costly Inn
F enumeration	Decent Inn
F enumeration	Cheap Inn

ST mealType	string
F enumeration	Kosher
F enumeration	Non-Vegetarian
F enumeration	Vegetarian

Figure A.2 Vocabulary of Travel_IVTS XSD



E Trip	(Trip)
E From	city
E To	city
E Mode	transport
A Type	string

E LocalCommute	(LocalCommute)
E Locality	city
A Type	string

E Flight	(Flight)
E Number	integer
E Source	airport
E Destination	airport
E Status	status
E Seat	integer
E Meal	mealType
A Type	string

E Carrier	airline
E Carrier	airlineName
A Mode	transport

E Hospitality	(Hospitality)
E Place	hotel
A Type	string

E Place	hotel
E Accomodation	hotelName
E Room	hotelRoom
E Availability	boolean

E Room	hotelRoom
A Smoking	boolean

E Meeting	(Meeting)
E Venue	meetingPlac
A Type	string

E Venue	meetingPlace
E Location	city
E Place	confCenters

Figure A.3 Abstractions in the Travel_IVTS XSD

Figure A.2 shows a subset of the vocabulary required to build the necessary abstractions of for the travel domain. Figure A.3 then presents a subset of our defined abstractions. Even in Figure A.3 observe how the domain vocabulary is used to build the abstractions (ex. Meeting has an element Venue). In our implementation the attribute "Type" denotes whether the element is a Concept or a Setting. Trip and Local Commute are Concepts and the rest are Settings.

Finally we present a subset of the IS and AS services defined in our travel IVTS. Figure A.4 shows the IS services and Figure A.5 shows the AS services. The IS services provide information for a set of affiliated Entries. For example, FlightInfo provides updates to a flight's status. The AS services allow action to be taken to modify existing Entries or create new ones. For example, AABook allows booking of a seat on an American Airlines flight.

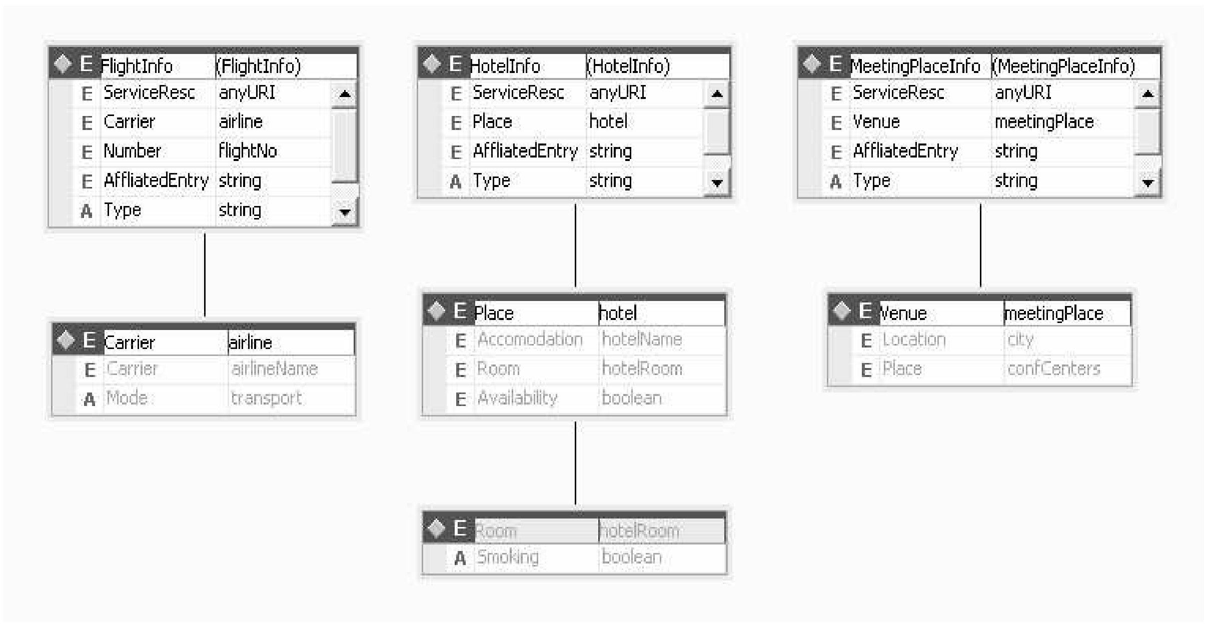


Figure A.4 IS services in the Travel_IVTS XSD

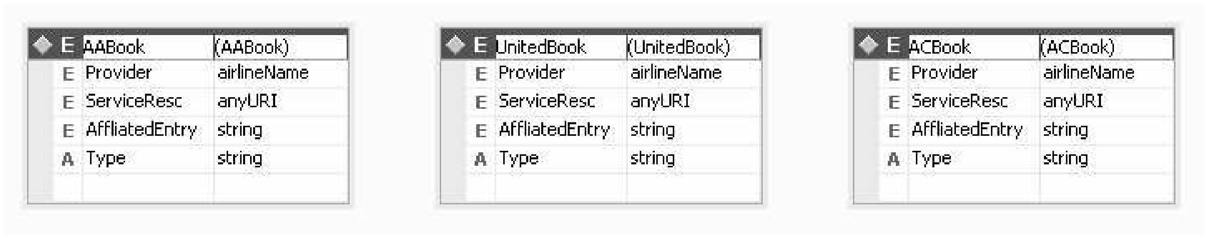


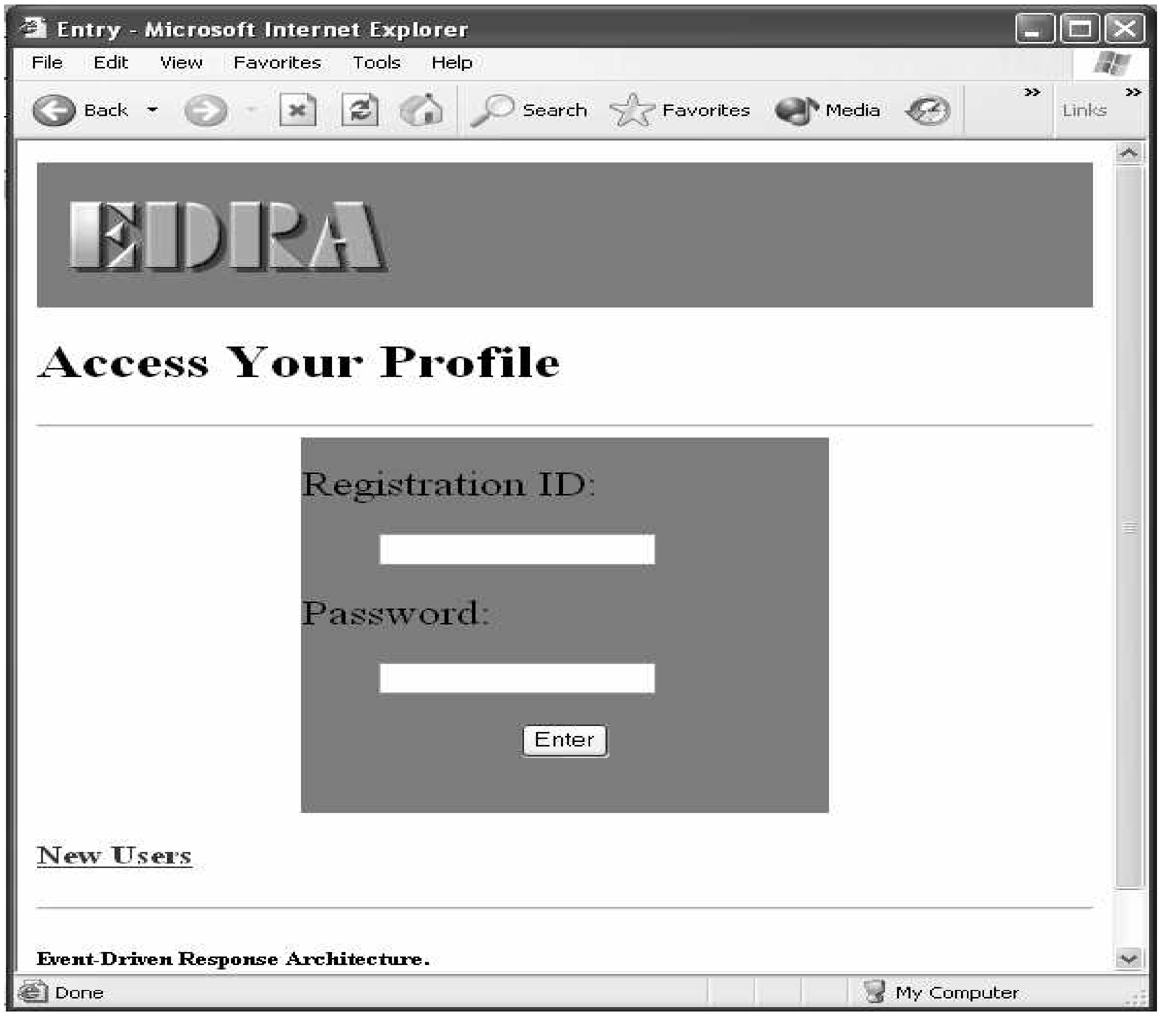
Figure A.5 AS services in the Travel_IVTS XSD

Though this IVTS knowledge specification describes a small subset of the travel industry, it demonstrates our approach for standardizing the data in an application domain to support automatic selection of relevant services based on clients' contexts. As mentioned in Chapter 3 the application domain and level of accuracy required will determine the development effort required.

Appendix B

Examples of Client Interaction with EDRA

An EDRA solution provides clients multiple interaction points as discussed in Chapter 3 and 4. Here we showcase an example of manual client interactions through an access device such as a PC. This also provides a good illustration of the form-based approach taken to keep clients' input consistent with the EDRA Data Model.


Entry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Links
EDRA

Access Your Profile

Registration ID:

Password:

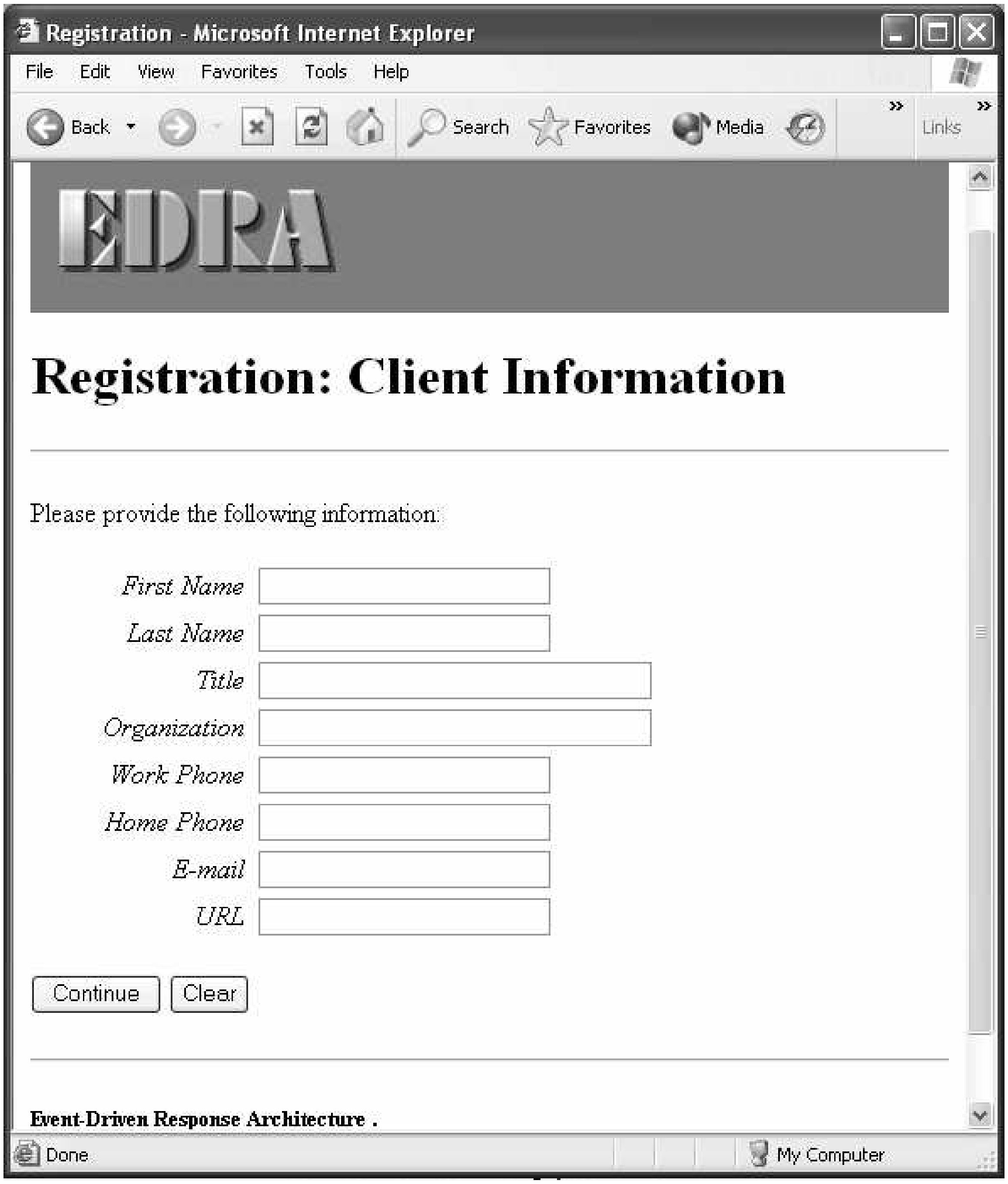
New Users

Event-Driven Response Architecture.

Figure B.1 Start Form

We begin with the Start Form which allows clients to either log into their EDRA service instance or register for a new one. In Figure B.1, we see that to log into an EDRA service instance the client needs to supply authentication information. To register for a new EDRA service instance, clients follow a link to the registration form.

The Figure B.2 depicts a simple registration form. This form requests the necessary information pertinent to the application domain of the EDRA solution.


Registration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Links
EDRA
Continue
Clear
Done
My Computer

Registration: Client Information

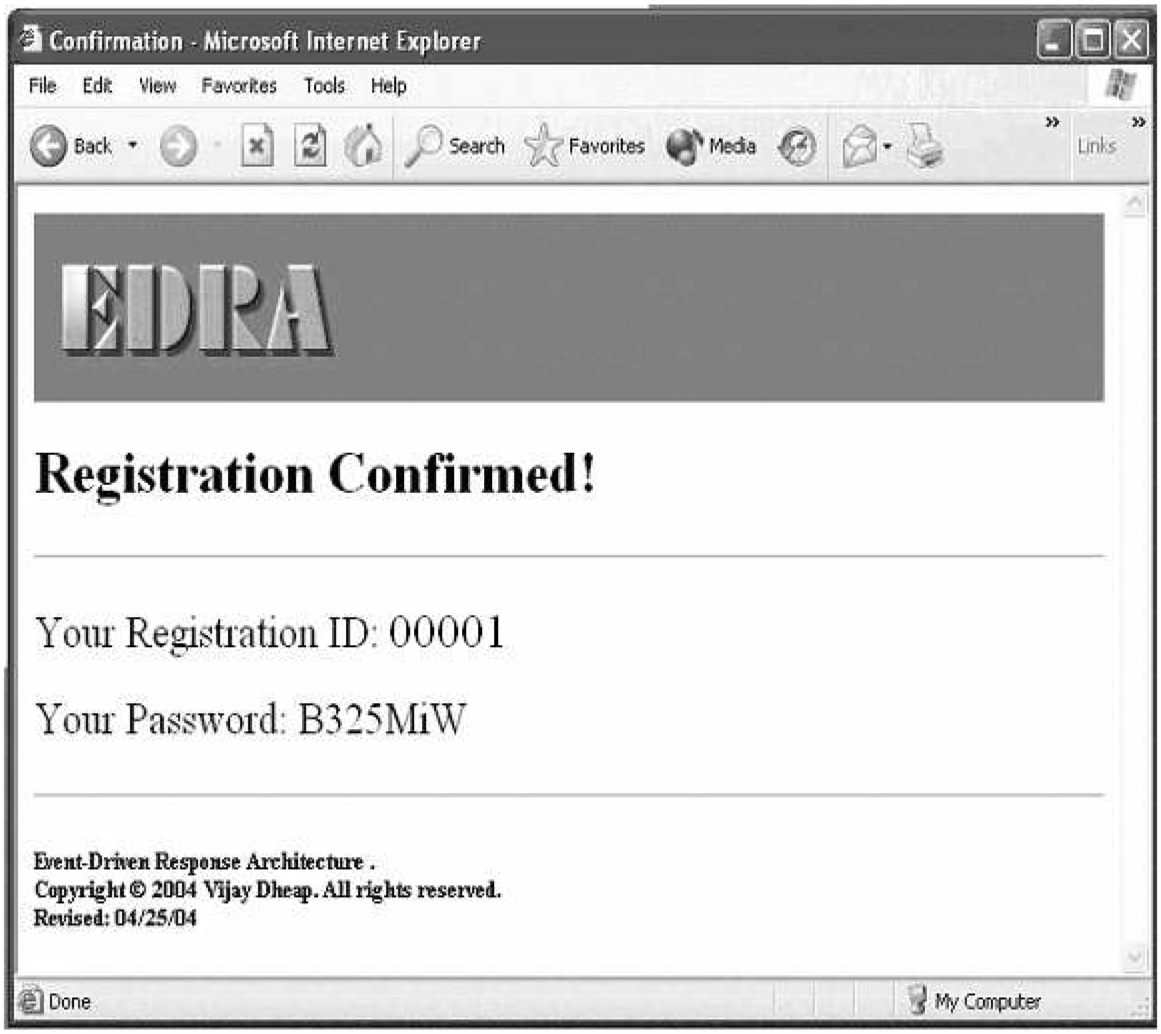
Please provide the following information:

First Name	
Last Name	
Title	
Organization	
Work Phone	
Home Phone	
E-mail	
URL	

Event-Driven Response Architecture .

Figure B.2 Registration Form

Once the client has entered the required information, he or she is given a registration ID and password. This can be used by the client for subsequent accesses to the EDRA solution to interact with their EDRA solution. Figure B.3 shows the Confirmation page.


Confirmation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links
EDRA
Done
My Computer

Registration Confirmed!

Your Registration ID: 00001

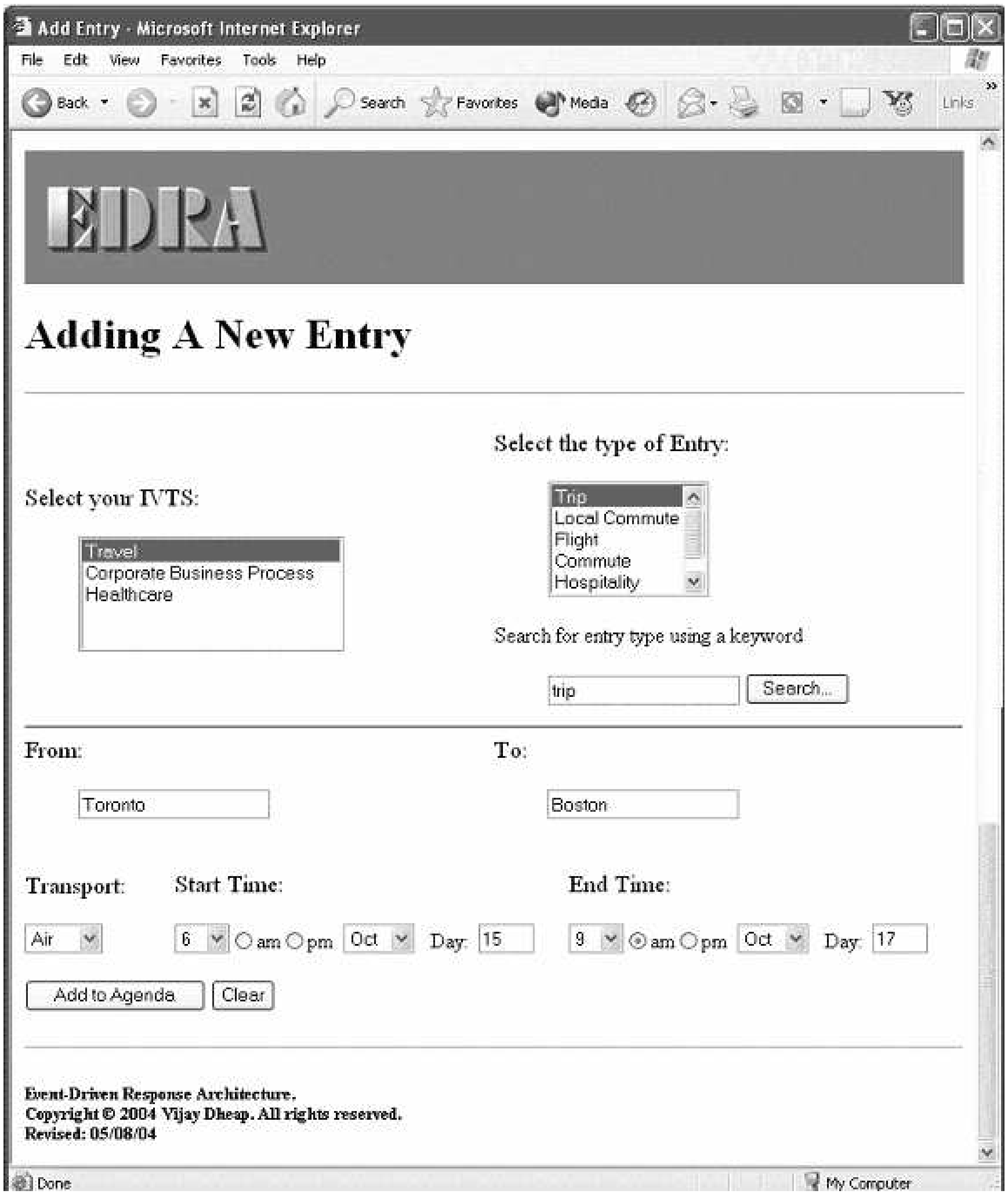
Your Password: B325MiW

Event-Driven Response Architecture .
Copyright © 2004 Vijay Dheap. All rights reserved.
Revised: 04/25/04

Figure B.3 Confirmation

For the rest of this example we will assume that the application domain of the EDRA solution is Travel. Our client is planning a trip from Toronto to Boston and wants to add the appropriate Entries of his trip into his EDRA service instance. We depict the parts of the interaction that occurs if the client chooses to use an access device to insert his travel plans.

The most general Entry in this scenario is “Trip”. This Entry is an abstraction of the client’s overall travel plan from Toronto to Boston. Figure B.4 demonstrates how the client will provide the necessary information for storing an Entry in the Agenda of the client’s EDRA service instance. This form based approach guarantees that the information is stored in an EDRA Data Model compatible format.


Add Entry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Links
EDRA

Adding A New Entry

Select the type of Entry:

Select your IVTS:

[✓ Travel]
Corporate Business Process
Healthcare

[✓ Trip]
Local Commute
Flight
Commute
Hospitality

Search for entry type using a keyword

trip Search...

From: Toronto

To: Boston

Transport: Air

Start Time: 6 am pm Oct Day: 15

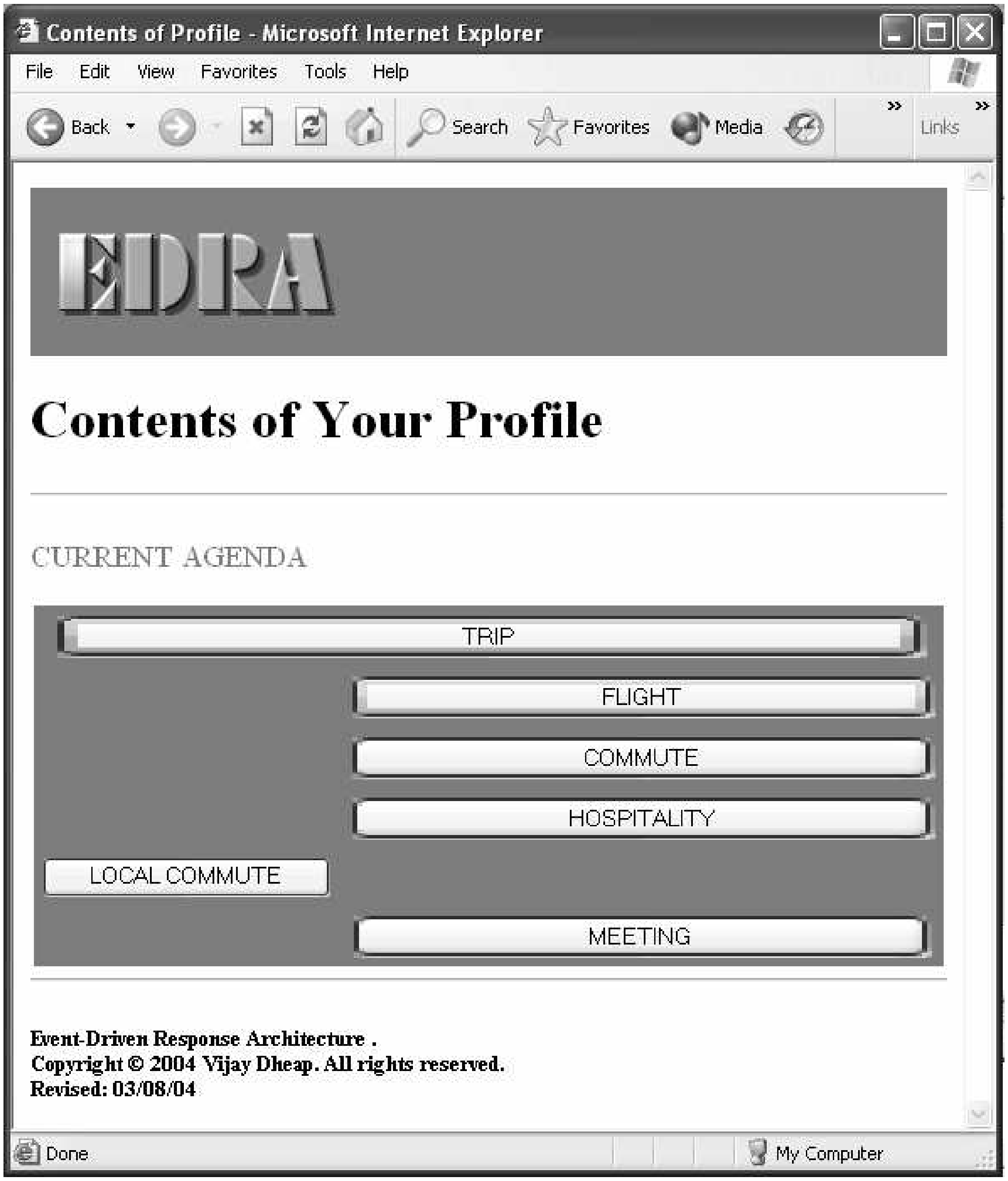
End Time: 9 [✓ am] pm Oct Day: 17

Add to Agenda Clear

Event-Driven Response Architecture.
Copyright © 2004 Vijay Dheap. All rights reserved.
Revised: 05/08/04

Figure B.4 Adding the Trip Entry

The new Entry is added into the agenda. Figure B.5 shows the contents of the client's Agenda after the client has entered all the Entries he wants EDRA to monitor and manage. The client may wish to edit or remove the Entry at a later point in time. If the client selects an Entry, the information the client previously provided is displayed, as shown in Figure B.6.


Contents of Profile - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Links
EDRA
Done My Computer

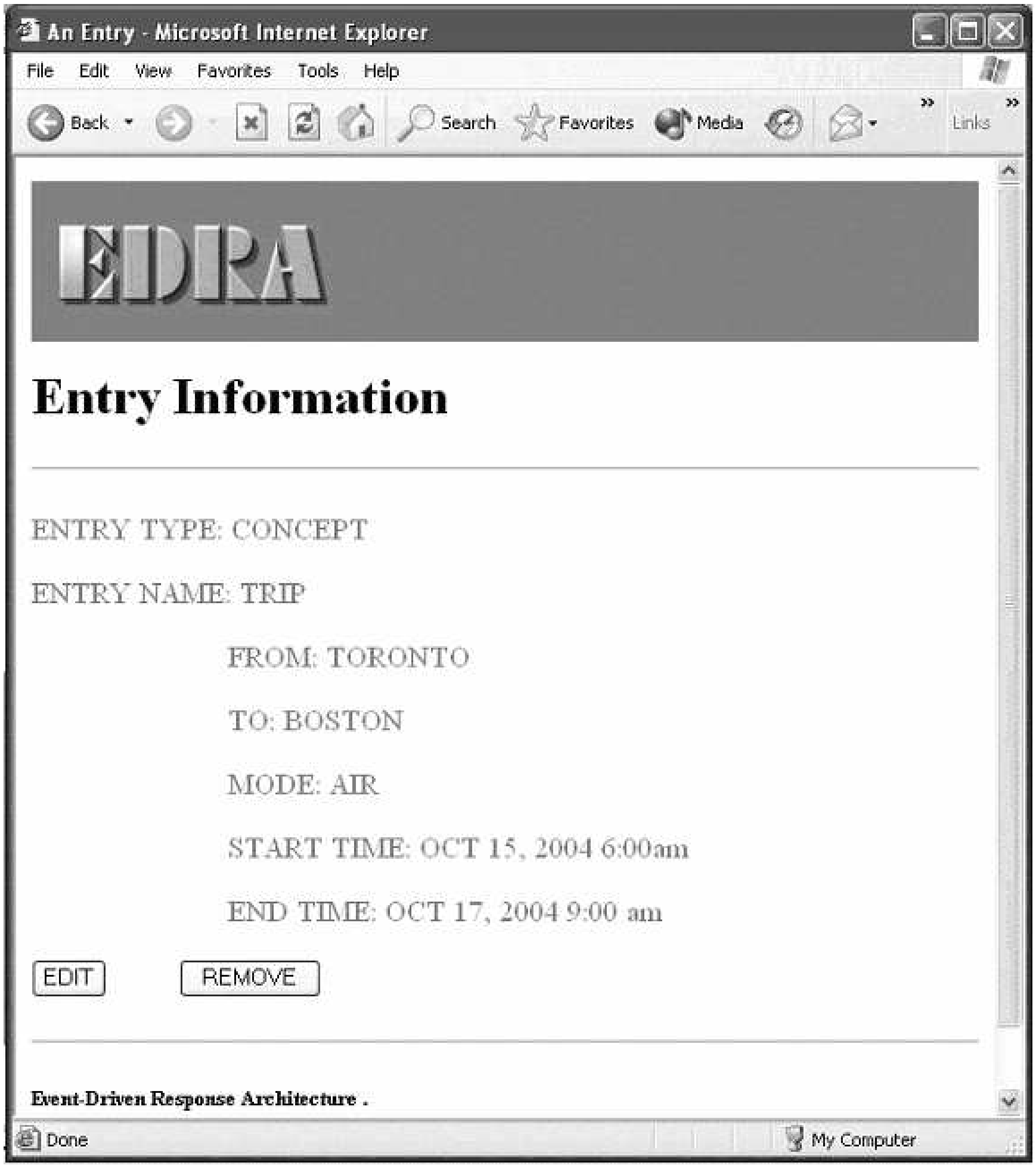
Contents of Your Profile

CURRENT AGENDA

TRIP
FLIGHT
COMMUTE
HOSPITALITY
LOCAL COMMUTE
MEETING

Event-Driven Response Architecture .
Copyright © 2004 Vijay Dheap. All rights reserved.
Revised: 03/08/04

Figure B.5 Client's Agenda


An Entry - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Links
EDRA

Entry Information

ENTRY TYPE: CONCEPT

ENTRY NAME: TRIP

FROM: TORONTO

TO: BOSTON

MODE: AIR

START TIME: OCT 15, 2004 6:00am

END TIME: OCT 17, 2004 9:00 am

Event-Driven Response Architecture .

Figure B.6 Selection of an Entry

A subset of the interactions the client has with the EDRA solution has been portrayed in this Appendix. The interactions we focused on are the client's manual accesses through an access device using a web client (e.g. Web browser). We provided this to improve understanding of the form-based approach to registration and adding Entries. In the process we have provided a glimpse of the minimal front-end that the client has to learn to adopt the EDRA solution. Simplification and ease of use to encourage adoption was an essential aspect of the framework's design.

Bibliography

[1] Arnstein, Larry et al. "Labscape: A Smart Environment for the Cell Biology Laboratory". IEEE Pervasive Computing Magazine vol. 1, no.3 (July-September 2002): pp 13-21.

[2] Barry & Associates. "Service Oriented Architecture (SOA) definition". Accessed: May 2004. <http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html>.

[3] Bieber, Guy and Jeff Carpenter. "Introduction into Service-Oriented Programming". Posted: 9 April 2001. Openwings. Accessed: May 2004. <http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf>.

[4] California Software Labs. "UPnP, Jini and Salutation – A look at some popular coordination frameworks for future networked devices". Accessed: May 2004. <http://www.cswl.com/whiteppr/tech/upnp.html>.

[5] Castro, Jaelson et al. "Towards Requirements-Driven Information Systems Engineering: The Tropos Project". Information Systems vol. 27, no. 6 (September 2002): pp 365-389.

[6] Cohen, Norman et al. "iQueue: A Pervasive Data Composition Framework". Proceedings on Mobile Data Management (January 2002): pp 146-153.

[7] Costa, Patricia Dockhorn. "Towards a Services Platform for Context Aware Applications". Telematics, University of Twente, Enschede. Netherlands. August 2003. Accessed: May 2004. <http://arch.cs.utwente.nl/assignments/thesis/ARCH-2003-04.pdf>.

[8] DARPA. "DARPA Agent Markup Language (DAML)". Accessed: May 2004. <http://www.daml.org/>.

[9] Eugster, P. Th. et al. "The Many Faces of Publish/Subscribe". ACM Computing Surveys vol. 35, no. 2 (June 2003): pp 114-131.

[10] Gamma, Erich et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[11] Harper, Philipp. "Informed Travelers = Fewer Flight Delays". Microsoft bCentral. Accessed: May 2004. <http://www.bcentral.com/articles/harper/129.asp>.

[12] Health Level Seven, Inc. "Health Level Seven (HL7)". Accessed: May 2004. <http://www.hl7.org/>.

[13] IBM Research. "Gryphon: Publish Subscribe Over Public Networks". IBM T.J. Watson Research Center. Accessed: May 2004. <http://www.research.ibm.com/gryphon/Gryphon/Gryphon-Overview.pdf>.

[14] Kentucky Department of Education. "Data Standardization". Posted: April 2004. Accessed: May 2004. <http://www.education.ky.gov/KDE/Administrative+Resources/Data+and+Research/Data+Standardization/default.htm>.

[15] Lei, Hui et al. "The Design and Applications of a Context Service". Mobile Computing and Communications Review vol. 6, no. 4. (October 2002): pp 45-55.

[16] MetaMatrix. "Enterprise Information Integration Whitepaper". Accessed: May 2004. <http://www.dmreview.com/whitepaper/wid339.pdf>.

[17] Microsoft Corp. "DCOM". Accessed: May 2004. <http://www.microsoft.com/com/tech/DCOM.asp>.

[18] OASIS. "HR-XML Consortium". Cover Pages. Accessed: May 2004. <http://xml.coverpages.org/hr-xml.html>.

[19] OASIS. "Universal Description, Discovery and Integration of Web Services (UDDI) Specification". Posted: July 2002. Accessed: May 2004. <http://www.uddi.org/specification.html>.

[20] Object Management Group. "History of CORBA". Accessed: May 2004. <http://www.omg.org/gettingstarted/history_of_corba.htm>.

[21] Object Management Group. "CORBA". Accessed: May 2004. <http://www.corba.org>.

[22] Open Financial Exchange (OFX). "About OFX". Accessed: May 2004. <http://www.ofx.net/ofx/ab_main.asp>.

[23] OOTips. "Model-View-Controller". Posted: May 1998. Accessed: May 2004. <http://ootips.org/mvc-pattern.html>.

[24] Paar, A. and W. Tichy. "Semantic Software Engineering Approaches for Automatic Service Lookup and Integration". Autonomic Computing Workshop – Active Middleware Series (June 2003): pp 103-111.

[25] Palmer, Sean B. "Semantic Web: An Introduction". Posted: September 2001. Accessed: May 2004. <http://infomesh.net/2001/swintro/>.

[26] Singh, Rahul. "RCal: An Autonomous Agent for Intelligent Distributed Meeting Scheduling". The Robotics Institute, Carnegie Mellon University. Nov 2003. Accessed: May 2004. <http://www.ri.cmu.edu/pub_files/pub4/singh_rahul_2003_1/singh_rahul_2003_1.pdf>.

[27] Sun Microsystems. "Java Blue Prints Model-View-Controller". Posted: 2002. Accessed: May 2004. <http://java.sun.com/blueprints/patterns/MVC-detailed.html>.

[28] Tanenbaum, Andrew S. and Maarten van Steen. Distributed Systems: Principles and Paradigms. Prentice Hall, 2002.

[29] Tosic, V. et al. "WSOL: Web Services Offerings Language". Proc. of the Workshop on Web Services, e-business, and Semantic Web, CaiSE'02 (May 2002).

[30] US House of Representatives – Transportation. 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>.

[31] Web Services Interoperability Organization (WS-I). "Resources and Guidelines for Web Services Interoperability". Accessed: May 2004. <http://www.ws-i.org/>.

[32] Winer, Dave. "XML-RPC Specification". Posted: June 2003. Userland XML-RPC. Accessed: May 2004. <http://www.xmlrpc.com/spec>.

[33] W3C Group. "Simple Object Access Protocol (SOAP) Version 1.2". Posted: June 2003. Accessed: May 2004. <http://www.w3.org/TR/soap12-part1/>.

[34] W3C Group. "Web Service Definition Language (WSDL) 1.1". Posted: March 2001. Accessed: May 2004. <http://www.w3.org/TR/wsdl>.