Modeling for E-Service Creation

Cécile Péraire [1] and Derek Coleman [2]

[1] SRI International, Menlo Park, CA
cecile@sdl.sri.com
[2] Hewlett-Packard Product Generation Consulting, Palo Alto, CA
derek_coleman@hp.com

Abstract. Today, the Internet provides a large set of mostly independent resources. In the near future, the Internet will provide e-services built by combining resources to achieve specific goals. This improvement will be possible thanks to new framework technologies, such as *E-Speak, Jini*, or the *Open Agent Architecture*. These technologies allow providers to create, compose, deploy, and advertise new e-services, and allow clients to discover and access these e-services. They permit the construction of brokers that dynamically discover and compose e-services to deliver the "best" solution. The emergence of new e-service based systems raises some new issues: What are the conceptual differences between e-service based systems and well-known distributed object-oriented systems? How can these differences be taken into account during the modeling process? The goal of this paper is to present a method, based on industry standard techniques, dedicated to modeling the architecture of e-service based systems.

Keywords. software architecture, e-business, e-components, e-services, frameworks, UML.

1 Introduction

If the pundits are to be believed, we are currently witnessing an evolution of Internet technologies that is going to bring electronic services (e-services) into everyday life and transform the way enterprises conduct electronic business (e-business). In the near future, a few years according to some authors [1, 2], almost everything may be an e-service, available from a Web site (e.g., e-service portal) or from anything having a microchip in it (e.g., car, phone, TV, or fridge), and delivered through a pervasive infrastructure. Today's monolithic systems will give way to dedicated e-services that can be dynamically combined in order to achieve high-level tasks. The evolution from today's distributed object-oriented systems to tomorrow's highly flexible e-service based systems will be possible thanks to new framework technologies, such as

- *E-Speak*, the service discovery and mediation framework developed by Hewlett-Packard [3],
- *Jini*, the service discovery framework developed by Sun Microsystems [4], or
- *The Open Agent Architecture*, the agent collaboration and delegation framework developed by SRI International [5].

These technologies (taken individually or in combination) provide mechanisms to create, compose, deploy, mediate, advertise, discover, and connect e-services. They permit the construction of service communities in which any e-service can be added, modified, or removed at run-time. They permit the building of brokers that dynamically discover the "best" e-service available at a time with respect to given quality criteria, and that compose e-services to deliver the "best" solution. Frameworks that mediate the communications between service provider and client facilitate the development of new monitoring services in order, for instance, to guarantee security in hostile unknown distributed dynamic environments.

The emergence of new e-service based systems raises some new issues: What are the conceptual differences between e-service based systems and well-known distributed object-oriented systems? How do these differences affect the modeling process? In this paper we show how to model e-services at the architectural level, using UML and standard object-oriented methodologies. We pay particular attention to the differences that e-services make to the modeling process.

The structure of the paper is as follows. First, the basic ideas of e-services are explained. Then, the main characteristics of some e-service frameworks such as *E-Speak*, *Jini*, and the *Open Agent Architecture* are described. This is followed by a case study to motivate and illustrate modeling-related issues. Finally, a method for modeling the architecture of e-service based systems is proposed, based on the UML notation and on a modeling process for e-service creation.

2 Introduction to E-services

This section explains the key concepts in e-services and why e-services constitute a new computational paradigm. We begin with a definition of an e-service:

- An **e-service** is some interaction offered to a user, across the Internet, that has meaning and economic value.

In the e-service community, the term *e-service* is often ambiguous because it is also used to refer to the piece of software providing the e-service. Where it reduces ambiguity, we use the term *e-component* for the software module.

- An **e-component** is a software module that provides one or more e-services. Thus, the e-services provided by an e-component constitute its interface.

In order to provide an e-service, an e-component usually has to interact with other e-components. In such interaction some e-component **delegates** or outsources a (sub) e-service to another e-component. In the outsourcing one component acts as the service *provider* for the other, the service *client* or *consumer*.

Thus, e-services are composed from the interactions of e-components acting as service providers and consumers. This is analogous to the way computation proceeds via communicating objects in a conventional object-oriented program. An important difference is that e-service interactions take place across the Internet and not within the confines of a single machine or LAN (Local Area Network).

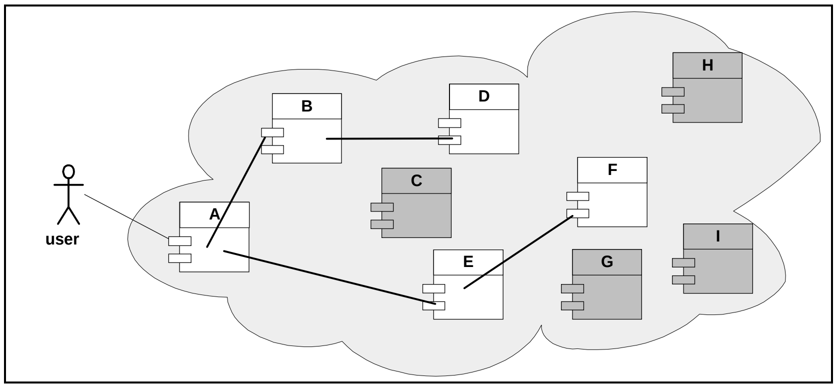


Fig. 1. E-Components Interacting to Provide an E-Service

Fig. 1 illustrates e-components *A*, *B*, *D*, *E*, and *F* collaborating as a network of service providers and consumers across the Internet to provide an e-service to a user. *A* is shown delegating some of its work to *B*, and so on. Other e-components, shaded gray, are not participating in this interaction.

E-components can **communicate** in a number of different ways. The communication can be by means of publish-and-subscribe, remote procedure calls, mailbox, or polling mechanisms. Publish-and-subscribe permits the loose coupling of e-components. It is a form of broadcast communication that allows e-components to be notified only of events in which they are interested. Remote procedure calls can be based either on specific interfaces or on generic interfaces that exchange XML [14] documents for instance. Mailboxes and polling mechanisms allow clients and providers to operate in a disconnected mode (they connect only periodically to check for available orders or responses).

It is important to realize that e-services are not just meant to provide services for humans or consumers. The e-services notion is quite general. Thus, the "stick person" actor symbol in Fig. 1 represents any service consumer that is connected to the Internet; indeed it can be an e-component just like the others. Thus, the e-service vision includes both business-to-consumer and business-to-business e-commerce.

E-components, like objects, encapsulate state behind an interface. In object-oriented systems, a basic motive for encapsulation is software reuse, i.e., the idea that systems can be composed from pre-existing objects that have been tried and tested in other systems. In the e-service context, the value of interfaces lies less in the ability to reuse e-components, but in the ability to *replace* them. The Internet provides the medium by which different providers of the same e-service can compete to be used in some interaction. Thus, if a service provider is inefficient, or too expensive to use, and if a cheaper or better provider is available, then the service consumer can dynamically switch to using the new e-component instead.

The mechanism by which service consumers can dynamically find service providers with respect to given criteria is called **discovery**. This is usually achieved by means of matching mechanisms, such as an attribute-based lookup, an interface-based lookup, or a Prolog unification.

Fig. 2 depicts the same e-service as in Fig. 1. E-component *B* has discovered service provider *C* and has decided to use it as a replacement for *D*. Likewise, *E* has discovered *G* and chosen to use it instead of *F*. However, it turns out that *G* delegates part of its work to some other e-component *I*.

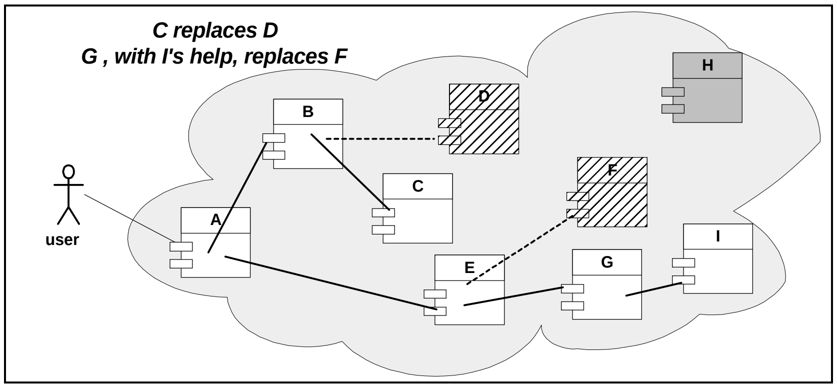


Fig. 2. Service Consumers can Dynamically Choose Different Service Providers

In order to be discovered, a service provider must **advertise** its services. This may be accomplished with the help of a service description that defines the services in a vocabulary shared by the community. The service description is provided by the e-component to an advertising service.

Thus, the e-service computational model is one in which e-services are delivered to users by impromptu collaborations between e-components which can be anywhere on the net. A choice to use a particular e-component at one moment may not be the right choice a moment later. New e-components, which are superior or more economic or more convenient to use, may have advertised themselves. Selecting the appropriate service provider with the "best" service available at a given time with respect to given quality criteria (e.g., price, availability, performance) is called **brokering**.

Brokering may involve negotiating with competing services. In many e-service systems there may be specialized components, called **brokers**. The job of a broker is to be a middleman that acts as a matchmaker between service consumers and service providers. In Fig. 1 and Fig. 2 e-component *B* is acting as a broker matchmaking the different service providers, *C* and *D*, with the service consumer *A*.

E-services may allow the communication between components to be **mediated**, i.e., monitored. This permits a *pay-per-use* economic model for using e-services. Instead of purchasing software outright as is the case today, e-services allow a model in which an e-service is paid for on a metered basis. Thus, e-service clients will have to pay only for what they consume.

Mediation also facilitates **security**, which is important for economic transactions over the potentially hostile environment of the Internet. E-service security is concerned with protecting the privacy of communicated information and authenticating the credentials of the e-components involved. In many circumstances *confidentiality* is important, allowing service providers and clients to remain mutually anonymous.

E-services need to be **heterogeneous**, i.e., e-components have to be allowed to run on different platforms and be written in different languages. E-services must also be **pervasive**. An e-service should be accessible from anywhere in the world from a variety of portable devices from desktops, laptops, PDAs, cell-phones and Internet-enabled smart devices such as automobiles or fridges. Pervasiveness is important because it broadens the range of e-services that can be provided. Portable devices mean that e-services can provide live updates such as changes in stock or commodity prices, and airline flight cancellations. Portability also equates to geographic distribution so that the e-services are not limited to desk-bound users but can interact with economic life in the real world as it happens, and wherever it might be.

An e-service should be able to be modified (e.g., its interface changed) without needing the modification and recompilation of its clients in order for them to be able to use the new version of the service. Similarly, an e-service should be capable of being removed without needing the modification and recompilation of its clients in order to use another equivalent service.

To conclude, e-services are distributed dynamic systems that work on the Internet. Although similar to distributed object-oriented systems, based on CORBA [12] and DCOM [13], e-services are fundamentally different because they have to be designed in the absence of full knowledge of the number and nature of their clients and providers and what else exists "out there" on the Internet. The vision of e-services can be summed up as "*let the Internet work for you, instead of you working the Internet*".

3 E-service Frameworks

This section introduces the software technologies that have been developed to support e-services. We briefly describe three current e-service frameworks, *E-Speak*, *Jini*, and the *Open Agent Architecture*, focussing on their key features and main differences.

3.1 E-Speak

This section presents the *E-Speak* framework technology. It is gathered from the documentation available at the *E-Speak* Web site [3].

The *E-Speak* framework provides mechanisms to create, compose, deploy, mediate, discover, and connect e-services. The aim is to form a distributed and dynamic community of e-services working together regardless of the technology platform on which they were built. The vision is to create an open service marketplace.

In order to join the community, services must register with an *E-Speak core*. To be discovered by other members of the community, services need to be advertised by providing their description to an advertising service (e.g., *E-Speak* local advertising service). These descriptions, based on attributes, must be expressed in a vocabulary shared by the community. When advertised, services can be discovered with an attribute-based lookup, and then accessed. A typical *E-Speak* session is illustrated in Fig. 3, with one core. Several cores, possibly located on different machines, may be interconnected, allowing a core to connect a client with any service of the community.

E-Speak supports a *Network Object Model* that allows clients to perform remote method invocations on remote objects using stubs, in a way similar to *Java* Remote Method Invocation. *E-Speak* also supports a publish-and-subscribe mechanism that allows a client or provider interested in a particular event to be notified each time this event occurs.

As shown in Fig. 3, *E-Speak* can mediate the communications between service provider and client. The mediation mechanism is used to implement security policies, i.e., authentication, access control, and confidentiality (virtual naming allows the service provider and client to remain mutually anonymous). Furthermore, mediation is a powerful mechanism to build new monitoring services, like metering or billing.

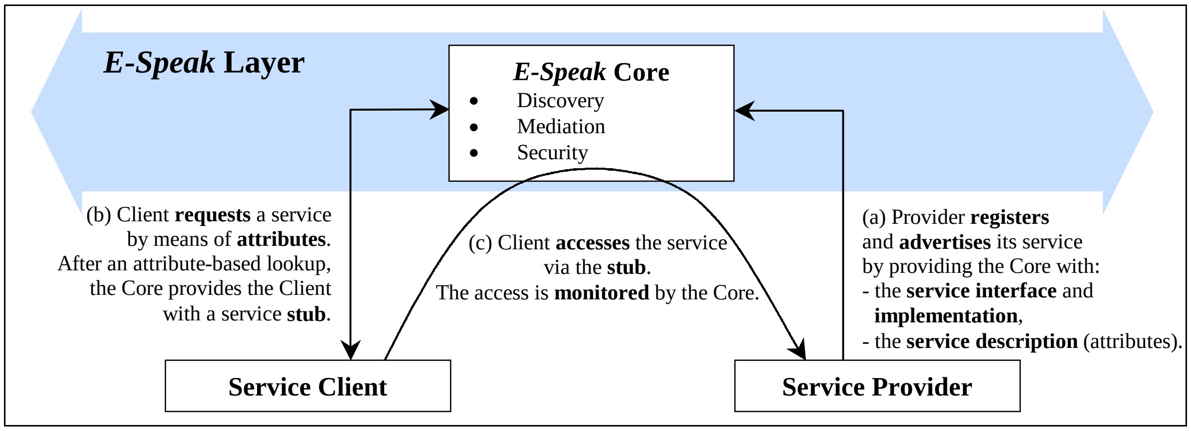

E-Speak Layer
E-Speak Core
Discovery
Mediation
Security
(b) Client requests a service by means of attributes. After an attribute-based lookup, the Core provides the Client with a service stub.
(c) Client accesses the service via the stub. The access is monitored by the Core.
(a) Provider registers and advertises its service by providing the Core with:
- the service interface and implementation,
- the service description (attributes).
Service Client
Service Provider

Fig. 3. A Typical *E-Speak* Session

The *E-Speak* library (version Beta 2.2) is written in *Java* and is available on the Windows NT, Linux, and HP-UX platforms. E-services are written in *Java* or in any other programming language as long as they are wrapped in *Java* components.

3.2 Jini

This section presents the *Jini Connection Technology*. It is gathered from the documentation available at the *Jini* Web site [4]. Additional information about *Jini* can be found in [6, 7].

Jini provides mechanisms that enable resources (hardware devices, software programs, or a combination of the two) to form a distributed and dynamic community of e-services. The vision is to create an impromptu network of self-managing devices (e.g., cell phones, printers, cameras).

In order to join the community, a service provider must discover a *lookup service* where it then uploads its service's object and attributes. The lookup service is a repository of services that acts as a switchboard to connect a client looking for a service (by means of the interface and attributes) with that service. Once the connection is made, the lookup service is not involved in any of the resulting interactions between client and service provider. Consequently, *Jini* does not supply mediation mechanisms between clients and providers.

The *Jini* framework is based on three protocols called *discover*, *join*, and *lookup*. A typical *Jini* session that illustrates these protocols is presented in Fig. 4, with one lookup service. Objects in a lookup service may include other lookup services, possibly located on different machines, and allowing hierarchical and distributed lookups.

The *Jini* technology is based on the *Java* programming language. Communication between devices can be accomplished using *Java* Remote Method Invocation (RMI) [8]. The discovery, join, and lookup protocols depend on the ability to move *Java* objects, including their code, between *Java* virtual machines. Also, *Jini* supports a publish-and-subscribe mechanism. An object may allow others objects to register interest in particular events and to be notified of the occurrence of these events.

In addition, *Jini* defines leasing and transaction mechanisms to provide resilience in dynamic networked environments. Leasing is used to detect when a service becomes unavailable and to perform a distributed garbage collection. When a service registers with a lookup service, it receives a lease that must be periodically renewed. If the lease is not renewed, then the lookup service removes the service from the list of the services offered. Transaction mechanisms are used to coordinate the actions and state changes of a group of distributed objects (fault-recovery).

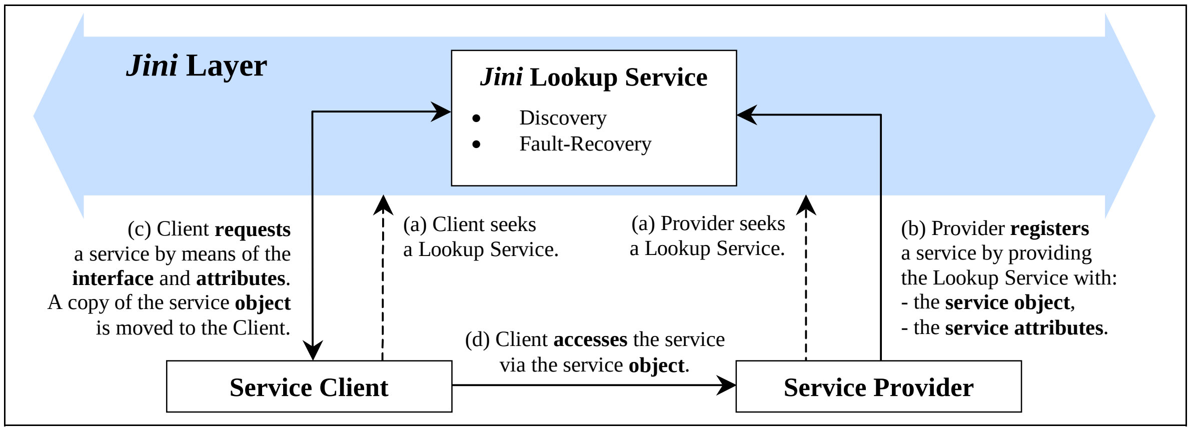

Jini Layer
Jini Lookup Service
Discovery
Fault-Recovery
(c) Client requests a service by means of the interface and attributes. A copy of the service object is moved to the Client.
(a) Client seeks a Lookup Service.
(a) Provider seeks a Lookup Service.
(b) Provider registers a service by providing the Lookup Service with: - the service object, - the service attributes.
(d) Client accesses the service via the service object.
Service Client
Service Provider

Fig. 4. A Typical *Jini* Session Illustrating the Discovery (a), Join (b), and Lookup (c) Protocols

The Jini library (version 1.1) is written in *Java* and is available on any platform having a *Java* Virtual Machine. E-services are thus supplied by *Java* components, or by any existing legacy application wrapped in a *Java* component.

3.3 The Open Agent Architecture

This section presents the *Open Agent Architecture* (*OAA*). It is gathered from the documentation available at the *OAA* Web site [5]. Additional information about the *OAA* can be found in [9, 10].

The *OAA* aims at building distributed and dynamic communities of agents, where multiple agents contribute services to the community, and where human users interact with the community via natural communication modes, such as speech, writing, or gesture.

Agents communicate among themselves using the *Inter-agent Communication Language* (*ICL*), which is a logic-based declarative language capable of representing natural language expressions. In order to join the community, a provider agent (service provider) must register its services (*ICL solvables*) with a *facilitator* agent. When requiring a service, a client agent (service client) submits to the facilitator an *ICL* expression (or a natural language expression that will be translated into an *ICL* expression by a dedicated agent) describing a high-level request. Thanks to the Prolog unification mechanism [11], the facilitator will make decisions about which agents are available and capable of handling sub-parts of the request, and will manage all agent interactions required to handle the complex query. A typical *OAA* session is illustrated in Fig. 5, with one facilitator agent. Several facilitator agents can be connected for scalability purposes.

Fig. 5 presents two service providers to illustrate the fact that most *OAA* requests from service clients are spread over multiple service providers that work either cooperatively or competitively on various aspects of the request. A request is expressed in terms of *what* is to be done rather than in terms of *how* it should be performed (control parameters specifying how the task should be performed are optional and separate from the task description itself). Thanks to this delegation model, *OAA* systems are very flexible. Indeed, an *OAA* request specifies a high-level task, and then the facilitator is responsible for dynamically finding a strategy to execute the task.

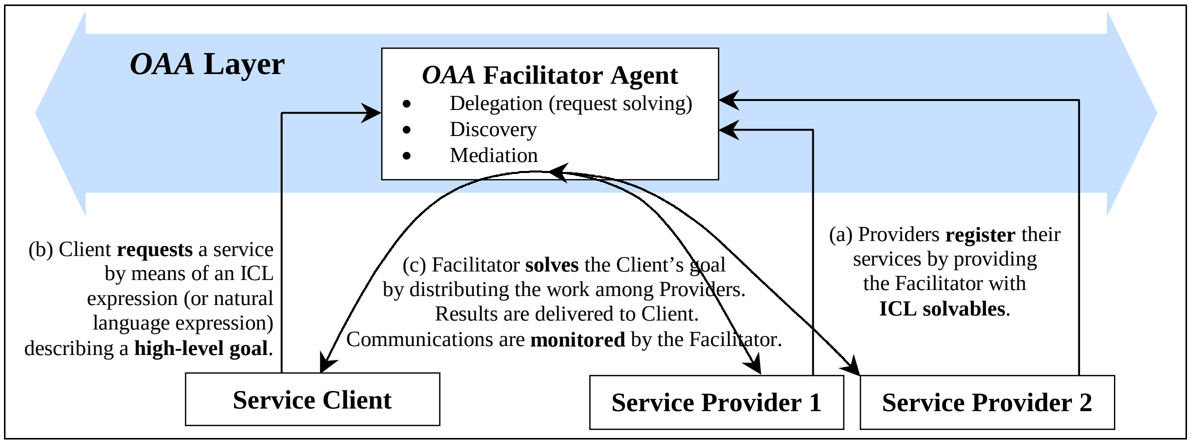

OAA Layer
OAA Facilitator Agent
Delegation (request solving)
Discovery
Mediation
(b) Client **requests** a service by means of an ICL expression (or natural language expression) describing a **high-level goal**.
(c) Facilitator **solves** the Client's goal by distributing the work among Providers. Results are delivered to Client. Communications are **monitored** by the Facilitator.
(a) Providers **register** their services by providing the Facilitator with **ICL solvables**.
Service Client
Service Provider 1
Service Provider 2

Fig. 5. A Typical *OAA* Session

As shown in Fig. 5, the *OAA* can mediate the communications between provider and client. In addition, the *OAA* supports a publish-and-subscribe mechanism, called *trigger*, by which an agent can specify an action to be taken when a condition is satisfied. Mediation and trigger are powerful mechanisms allowing the building of new monitoring agents.

The *OAA* libraries (version 2.0) are written in *Java*, *Prolog*, and *WebL*, and are available on the Solaris and Windows 9x/NT platforms. Agents are thus written in those programming languages, and they can be interfaced with any existing legacy application.

4 Travel Agency Case Study

This section introduces a travel agency case study, which will motivate our models later in the paper. This case study is inspired by a scenario, called *"Anticipating the unforeseeable"*, taken from the Hewlett-Packard e-service Web site [15].

Imagine a user who wants to book a travel package via the Internet, including flight, taxi, and hotel. What does he do today? From his palm computer for instance, he uses his favorite browser to find and access the portal[1] of an online travel agency. Then, he provides the travel agency with his travel specification. Later on, he books the corresponding travel by providing his credit card number. He will receive some e-tickets by mail a few days later. Fig. 6 illustrates the actions executed by the user (actions are arrows in the figure) from travel booking to travel ending.

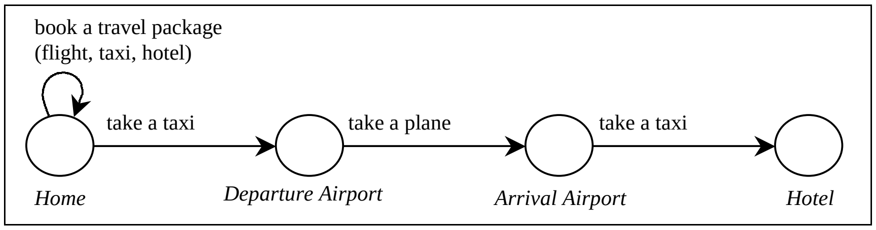


Fig. 6. Traveler Transition System for the "Happy Case"

Imagine that tomorrow, the travel agency will be an e-service based system composed of dozens of e-services collaborating to maximize the efficiency of the whole business. What will be the differences from the user's point of view? Perhaps there will be no differences at all! The user would continue to book his travel as before, using his favorite browser and online travel agency[2], and the transition system presented in Fig. 6 would apply. So what? What are the advantages of e-service based systems?

Consider the following scenario: *There is nothing like leaving on a business trip the day before Thanksgiving. The hapless executive heads for the airport. While he is en route, all hell breaks loose in Salt Lake City. A blizzard grounds his connecting flight to LaGuardia. All other flights to New York City are overbooked. And by the way, New York City's taxi drivers have just gone on strike.*

What may happen to the executive today? In order to minimize his delay and problems, the executive may book another flight going to the nearest airport (e.g., Newark), inform his hotel about late arrival, rent a car and find directions from the airport to his hotel, and so on. An example of actions to be executed by the executive is illustrated in Fig. 7.

What may happen to the executive tomorrow, when the travel agency will be an e-service based system? By the time the executive would get to the airport, the Internet would go to work, turning a near disaster into a minor delay without interventions of the executive, as illustrated in Fig. 8, in which the dotted arrows correspond to actions executed by the e-service system.

Facing such a situation could be part of the requirements given to an architect in order to build a travel system able to "anticipate the unforeseeable". This is the assumption we make in this paper, in which the travel system is used to illustrate the modeling of the architecture of e-service based systems.

[1] A portal is a Web site that attempts to be an attractive starting point for accessing the Web. It offers a broad array of resources and services, such as e-mail, forums, search engines, and on-line shopping malls.
[2] E-services will be available from Web sites, or from anything having a microchip in it (e.g., pager, watch, pacemaker, sailboat).

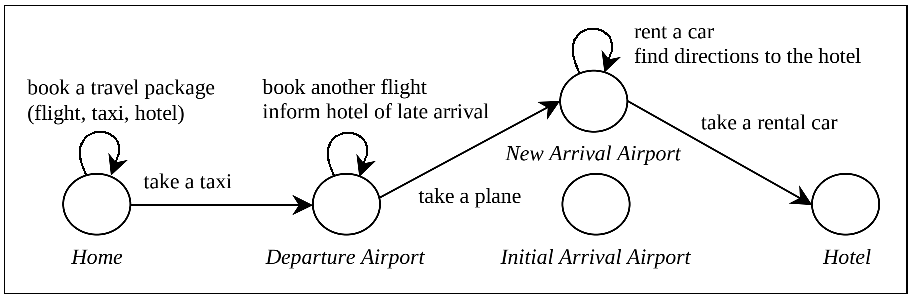


Fig. 7. Traveler Transition System for the "Crisis Case" - Today

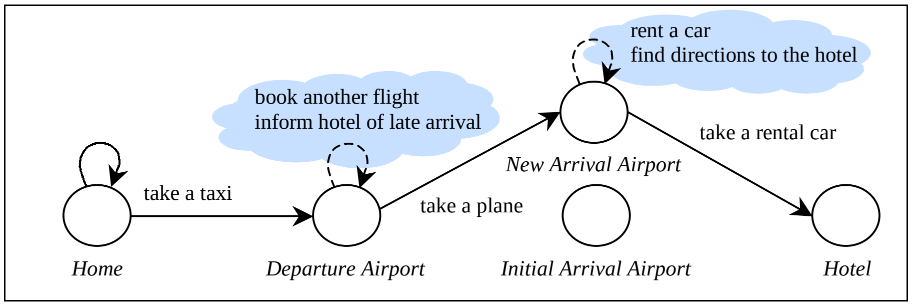


Fig. 8. Traveler Transition System for the "Crisis Case" – Tomorrow

5 How to Architect E-Services

Software architecture focuses on the structure and properties of the overall system rather than on the choice of computation algorithms. According to Bass et al [16], the software architecture of a system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. The IEEE recommendation [17] defines an architecture as the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution. As stated in [18] software architectures are important because they represent the single abstraction for understanding the structure of a system and form the basis for a shared understanding of a system and all its stakeholders (product teams, hardware and marketing engineers, senior management, and external partners).

In the context of e-service based systems, we define an architecture in terms of *structure* and *properties* as illustrated in Fig. 9. The structure consists of the system's *topology*, i.e., how e-components relate to each other, and of the *e-component* and *e-service* specification. The properties express the system's *dynamic behaviors* from an external and internal point of view.

E-service based systems are structured around the notion of e-service. Then, an e-service can be realized by any communication style supported by the e-service framework (e.g., remote procedure calls, publish-and-subscribe, mailbox). Consequently, to allow architects to reason about the architecture of e-service based systems at a high level of abstraction it is necessary to introduce *e-services as first class entities*. This aspect will be widely illustrated by the modeling of the travel agency architecture.

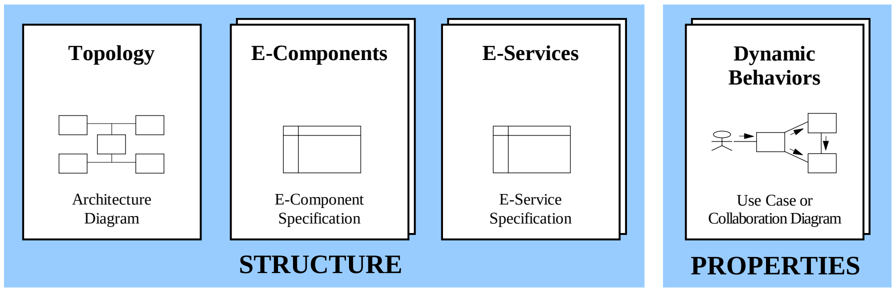


Fig. 9. E-Service Based System Architecture

The notation for architectural description is based on UML [19, 20]. Our aim is to use as much as possible the standard UML models in order to remain close to current best practice. Nevertheless, minor extensions are necessary to express some specific e-service mechanisms. Furthermore, the system's structure and properties are documented as proposed in [18], which conforms to the IEEE Recommendation for Architectural Description [17]. As illustrated in Fig. 9, the system's topology is documented using an architecture diagram, e-components and e-services are documented using specification templates, and dynamic behaviors are documented using collaboration diagrams or use cases.

In order to devise a conceptual architecture for e-services we propose the following steps:

- **Initial Static Architecture.** This is a eureka-step in which the architect proposes a first cut at the architecture based on experience. The architecture can be validated by running scenarios against the architecture as is recommended by the Software Architecture Assessment Method (SAAM) developed by Bass et al [16]. We call this a "static architecture" because it ignores most of the dynamic issues inherent in e-services.
- **Advertising and Discovery.** Use the economic context of the e-services to define what, when, and how to advertise and discover.
- **Communication Styles**. Make the e-services more concrete by defining the communication styles (e.g., remote procedure calls, publish-and-subscribe) used to provide and outsource e-services.
- **Non-Functional Requirements.** Modify the architecture to take non-functional requirements, like security and scalability, into account.
- **Choice of E-service Framework.** Modify the architecture to take into account the particular semantics of the e-service framework to be used.

The following sections consider each step of the process in turn.

5.1 Initial static Architecture

A topology diagram showing the e-components and their e-services documents the architecture. The topology is supplemented with specifications of each e-component and e-service.

5.1.1 Topology

Creating a topology is a eureka-step that comes from the experience of the architect. The topology of the travel agency e-service system is illustrated in Fig. 10. The figure presents a very high level view of the system showing the e-components together with the e-services that they provide and require. It shows that the e-component E-TravelAgency delegates e-services supplied by E-TravelBroker, which in turn delegates e-services supplied by E-Airline, E-Hotel, E-Transport, and E-CarAgency. The part of the figure related to

E-GPS illustrates that a given e-service set (*GPS-1Services*) can be delegated by different e-components (E-Transport and E-CarAgency), and that a given e-component (E-GPS) can offer different e-service sets (*GPS-1Services*, *GPS-2Services* and *GPS-3Services*). This allows an e-component to provide clients with different sets of services having distinct purposes. The specification of some e-components and e-services is presented in the two next sections.

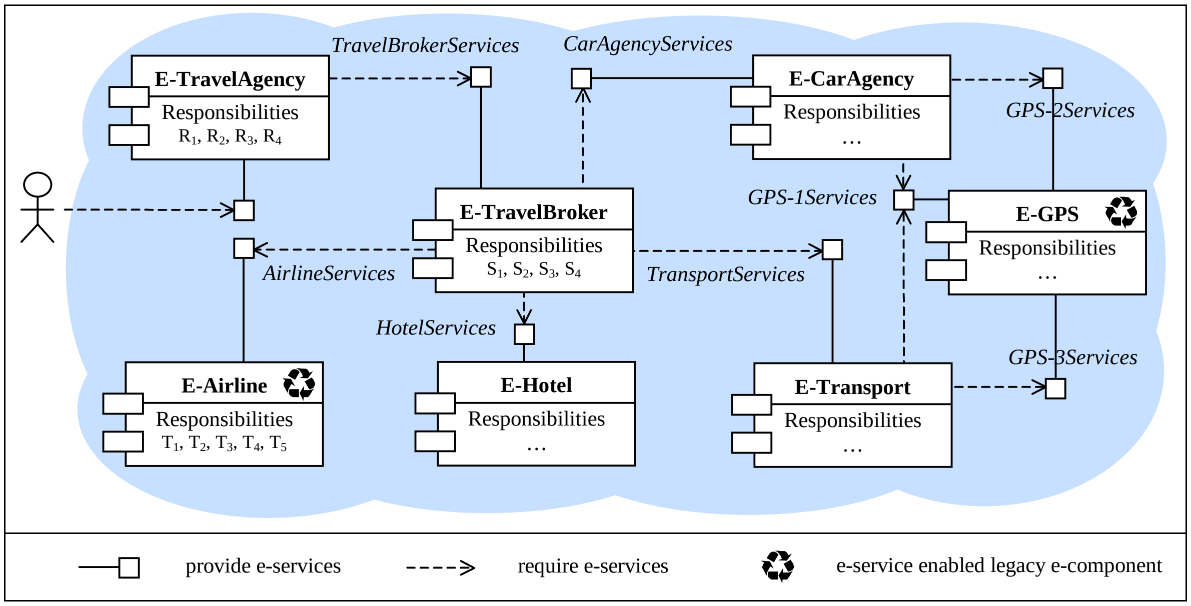


Fig. 10. Architecture Diagram – E-Service Level

The diagram uses a slight extension of UML. An e-component is modeled using a UML component, i.e., a rectangle with tabs and a special compartment showing its responsibilities. In UML, a component is defined as a *physical* and *replaceable* part of a system that conforms to and provides the realization of a set of interfaces [20]. For modeling e-service based systems, we substitute *e-service* for *interface* as the glue that binds components together. As we discuss in Section 5.3, an e-service can be realized by any communication style supported by the e-service framework (e.g., remote procedure calls, publish-and-subscribe, mailbox, polling). In order to model e-services, the following UML compatible notations are introduced:

- A line ending with a white square expresses that the e-component (service provider) provides e-services.
- A dotted arrow (i.e., a UML dependency relationship) expresses that the e-component (service client) requires e-services.
- The recyclable symbol indicates that an e-component is an e-service enabled legacy application, i.e., an existing legacy application wrapped into an e-component.

5.1.2 E-Component Specification

Each e-component is described using an e-component specification template (see Table 10 of the Appendix). Briefly, an e-component specification describes an e-component in terms of its responsibilities.

The e-component specification for the E-TravelAgency is presented in Table 1. E-TravelAgency allows the travel agency to switch on/off the travel system by means of a GUI for instance. The traveler may use a portal to access the services offered by E-TravelAgency.

Table 1. E-TravelAgency Specification

E-Component	E-TravelAgency
Responsibilities	• R_1: Allow the traveler to rapidly explore travel itineraries, given a travel specification (possibly incomplete) and some prioritized constraints. • R_2: Allow the traveler to book travel packages at discount price. • R_3: Allow the traveler to make trouble-free journeys. In case of trouble a modified itinerary is proposed to the traveler. • R_4: Maintain a database of travelers and travel package identifiers.

While E-TravelAgency is responsible for proposing services to travelers, E-TravelBroker is responsible for processing these services. The e-component specification for the E-TravelBroker is presented in Table 2. E-TravelBroker locates the best travel suppliers for the itinerary; it also mediates all communication between the chosen airline, hotel, and so on, and the travel agency. This allows it to monitor the state of each itinerary that has been booked and do any necessary rebooking. This also allows it to take a small pay-per-use charge for all transactions, including those that occur if the airline announces a delay to a flight.

Table 2. E-TravelBroker Specification

E-Component	E-TravelBroker
Responsibilities	• S_1: Compute all possible travel itineraries and prices, given a travel specification (possibly incomplete) and some prioritized constraints. • S_2: Process order bookings. • S_3: Monitor the booked travel itineraries and, in case of trouble, compute a modified itinerary. • S_4: Maintain a database of travelers and itineraries.

Table 3 shows the e-component specification for the E-Airline. Again, E-Airline may allow the airline to access the airline system by means of a GUI.

Table 3. E-Airline Specification

E-Component	E-Airline
Responsibilities	• T_1: Allow the airline to modify the database of flights, seats, and travelers. • $T_{2:}$ Allow clients (e.g., E-TravelBroker) to search the seat database. • T_3: Allow clients (e.g., E-TravelBroker) to book a seat. • T_4: Inform interested parties about flight delays or cancellations. • T_5: Maintain a database of flights, seats, and travelers.

5.1.3 E-Service Specification

An e-service is a meaningful capability that can be offered for any valuable interaction. E-services working for the same goal are grouped into an e-service set. An e-service set specification describes the high level goals, and lists the e-services constituting the set (see Table 11 of the Appendix).

The e-service set specification for *TravelBrokerServices* is presented in Table 4. E-services A_1 to A_4 are grouped because they are working for a same goal, that is, the selling of crisis-handling travel itineraries.

Table 4. *TravelBrokerServices* Specification

E-Service Set	*TravelBrokerServices*
Description	Services provided by E-TravelBroker to assist the clients (e.g., E-TravelAgency) in selling crisis-handling travel itineraries.
E-Services	• A_1: Search for travel itineraries. • A_2: Book a crisis-handling itinerary. • A_3: Send information about the traveler's ability to execute the schedule (e.g., feedback on a modified itinerary, delay) to the travel system. • A_4: Receive relevant information about travel itineraries (e.g., new schedule).

Similarly, the e-service set specification for *AirlineServices* is presented in Table 5. E-services B_1 to B_4 are working for the same goal, that is, the selling of flight tickets in the context of crisis-handling travel itineraries.

Table 5. *AirlineServices* Specification

E-Service Set	*AirlineServices*
Description	Service set provided by E-Airline to allow the consumers (e.g., E-TravelBroker) to purchase flight tickets in the context of crisis-handling travel itineraries.
E-Services	• B_1: Search the seat database. • B_2: Book a seat. • B_3: Send information about the traveler's ability to execute the schedule (e.g., delay) to the airline system. • B_4: Receive relevant information about flights (e.g., delay, cancellation).

This section has presented the static architecture of the travel system, i.e., the system's topology together with the e-component and e-service specification. Despite a few notations specific to e-services, the structure of the travel system could be the one of any distributed object-oriented system. The next sections present a method dedicated to modeling the highly dynamic behaviors of e-service based systems.

5.2 Advertising-and-Discovery

Advertising-and-discovery is a key mechanism that permits the construction of communities of e-components that collaborate without being tightly bounded like objects in distributed object-oriented systems.

5.2.1 What to advertise and discover?

In order to find out *what* e-services should be discovered and thus advertised, the architect defines the type of each e-service set from the client point of view. For this purpose, we use the notions of *commodity*, *replaceable*, *mission-critical*, and *identified* service defined as follows.

Commodity service: Service provided by any of a number of providers. Replacing one provider with another does not affect the system functionality. The productivity of the system is not reduced if the service is unavailable for a period of time. The service discovery process is continuous (each service access is preceded by a discovery phase).

Replaceable service: Service provided most of time by one or several preferred providers. Replacing one provider with another should not affect the system functionality. The productivity of the system is not severely reduced if the service is unavailable for a period of time. The service discovery process is discrete; its frequency varies over time.

Mission-critical service: Service always provided by one specific provider. Replacing one provider with another severely affects the system functionality. The productivity of the system is severely reduced if the service is unavailable for a period of time. The service discovery process is discrete; its frequency should be minimized.

Identified service: Service provided by a well-known provider. The service access is based on the provider identifier. There is no discovery process.

The type of each e-service set from the client point of view can be graphically rendered using UML stereotypes as illustrated in Fig. 11.

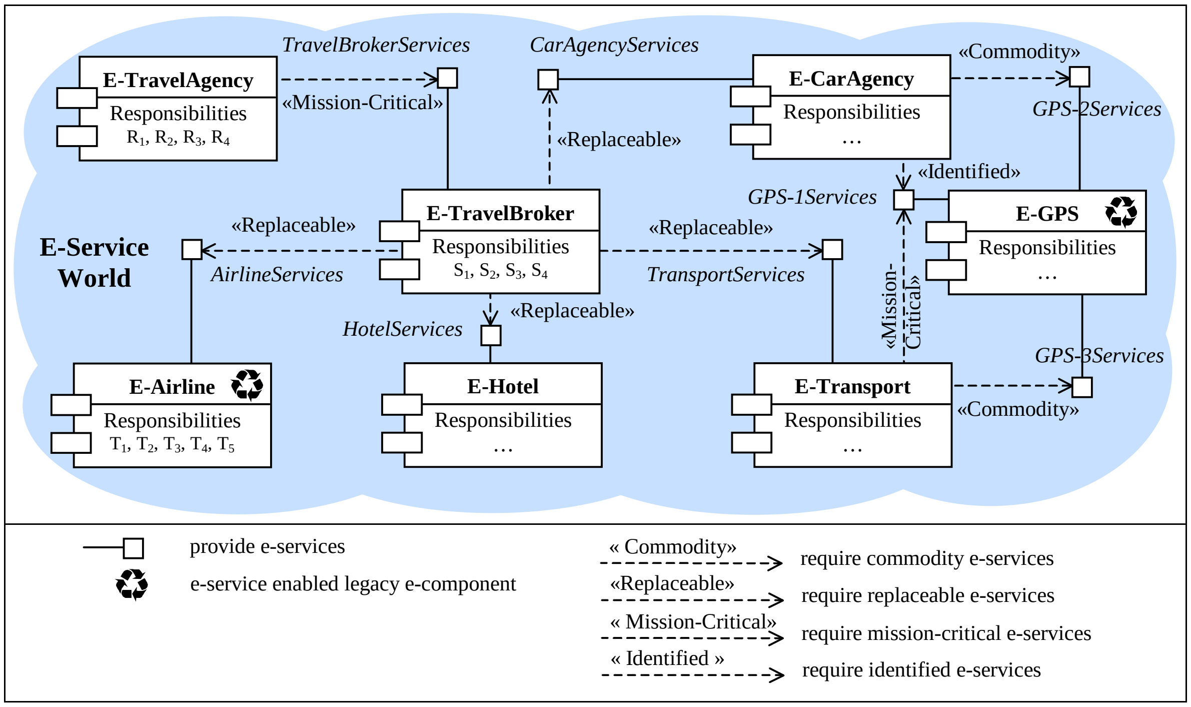


Fig. 11. Architecture Diagram – E-Service Level with Discovery

Fig. 11 shows that e-services supplied by *TravelBrokerServices* are *mission-critical* for E-TravelAgency. Indeed, the services provided by E-TravelBroker are crucial for E-TravelAgency. If these services are unavailable for a period of time, E-TravelAgency is unable to supply crisis-handling itineraries during this period. E-services offered by *AirlineServices*, *HotelServices*, *TransportServices*, and *CarAgencyServices* are *replaceable* for E-TravelBroker. Indeed, E-TravelBroker realizes some transactions with a group of preferred partners that change over time. E-services supplied by *GPS-2Services* and *GPS-3Services* are *commodity* for E-CarAgency and E-Transport. Indeed, positions can be supplied by any E-GPS. In addition, the system productivity will not be reduced if some positions are unavailable for a period of time. E-services supplied by *GPS-1Services* are *identified* for E-CarAgency and *mission-critical* for E-Transport. This shows that a given e-service set can have different types for different clients. Finally, from Fig. 11, the architect deduces that all the e-service sets of the travel system need to be discovered and thus advertised.

5.2.2 When to advertise and discover?

Once the e-services that should be advertised and discovered are defined as presented above, the architect specifies *when* these e-services are advertised and discovered.

In this paper, we assume that all the e-components advertise their e-services when they join the e-service community. Nevertheless, if it is not the case, the techniques presented below, for specifying when an e-service set is discovered, can be used for specifying when an e-service set is advertised.

For the discovery part, the architect draws the interactions between e-components and specifies which interactions must be preceded by a discovery phase. For this purpose, the UML collaboration diagrams are utilized. A collaboration diagram shows a set of objects (instances of e-components in our case), links among those objects, and messages sent and received by those objects. A message is a request or a result sent in the context of an e-service. This context is shown as a UML note on the collaboration diagram.

The collaboration diagram for the e-service *SearchForTravelItineraries*, offered to the traveler by the system, is presented in Fig. 12. This e-service searches for travel itineraries, starting from a (possibly incomplete) travel specification given by the traveler. We assume that the travel specification involves the search for a flight seat, a hotel room, and a rental car.

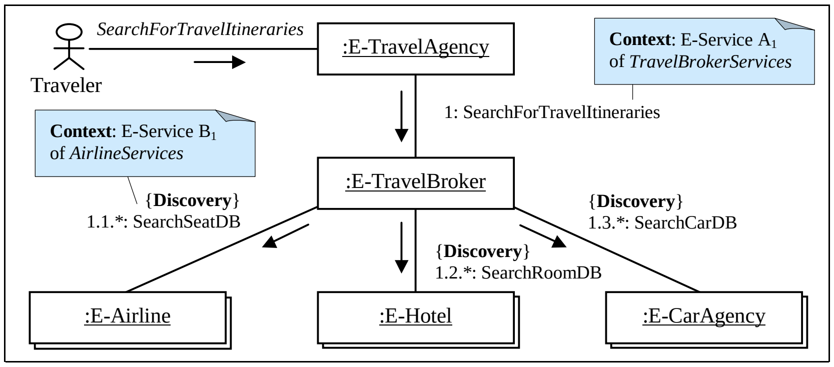

SearchForTravelItineraries
Traveler
:E-TravelAgency
Context: E-Service A1 of TravelBrokerServices
1: SearchForTravelItineraries
Context: E-Service B1 of AirlineServices
:E-TravelBroker
{Discovery} 1.1.*: SearchSeatDB
{Discovery} 1.2.*: SearchRoomDB
{Discovery} 1.3.*: SearchCarDB
:E-Airline
:E-Hotel
:E-CarAgency

Fig. 12. Collaboration Diagram for the E-Service *SearchForTravelItineraries*

For each request of the collaboration diagram, sent by an e-component *A* to an e-component *B*, the architect asks the question: *"Does A need to discover the e-services offered by B?"*

The answer is *yes* for *commodity* services, and *no* for *identified* services. For *replaceable* and *mission-critical* services, the question is decided by evaluating issues such as *"Are there likely to be multiple service providers for this e-service?"* or *"Can the e-service provider be replaced on the fly without any prior warning?"*[3]

If the answer is *yes*, the collaboration diagram is supplemented with the constraint {Discovery} preceding the request name. Such a constraint means that the request must be preceded by a discovery phase in which *A* discovers the e-services offered by *B*.

If the answer is *no*, the architect has to make sure that *A* already knows the identifier of *B* (i.e., *A* requires *identified* services from *B*, or *A* has already discovered *B*).

[3] The bank service provider in a bank processing transaction, such as deposit money, cannot be replaced without first moving bank accounts!

Step 1.1 of Fig. 12 shows that the answer to the question "*Does E-TravelBroker need to discover the AirlineServices offered by E-Airline?*" is yes. Indeed, *AirlineServices* are *replaceable* services for E-TravelBroker and it is the beginning of a new session. On the contrary, in step 1, E-TravelAgency does not have to discover the *TravelBrokerServices* offered by E-TravelBroker. Since *TravelBrokerServices* are *mission-critical* services for E-TravelAgency, the discovery phase has already occurred.

Because the details of *how* to advertise and discover e-services are framework specific, we return to this issue in last step of the process (see Section 5.5.1).

5.3 Communication Styles

After deciding which e-services have to be advertised and discovered, it is necessary to decide on the type of communication to be employed. This step thus makes the e-component interactions more concrete. Communication styles include any mechanism supported by the e-service framework, such as remote procedure calls (i.e., interfaces) or publish-and-subscribe (i.e., event broadcasting).

The publish-and-subscribe mechanism is an event-based communication style that allows e-components to broadcast important information to all the interested parties without establishing explicit connections between e-components. Each e-service framework proposes its own version of the mechanism. However, the differences are slight and do not affect the modeling process.

Publish-and-subscribe allows e-components to publish information on the one hand, and to subscribe to information on the other hand. This is done by means of events. The event is sent via the e-service layer (or through some third party), from the e-component that publishes the event to all the e-components (possibly located on different machines) that subscribe to the event. The reception of an event may allow the subscriber to access additional information associated with the event.

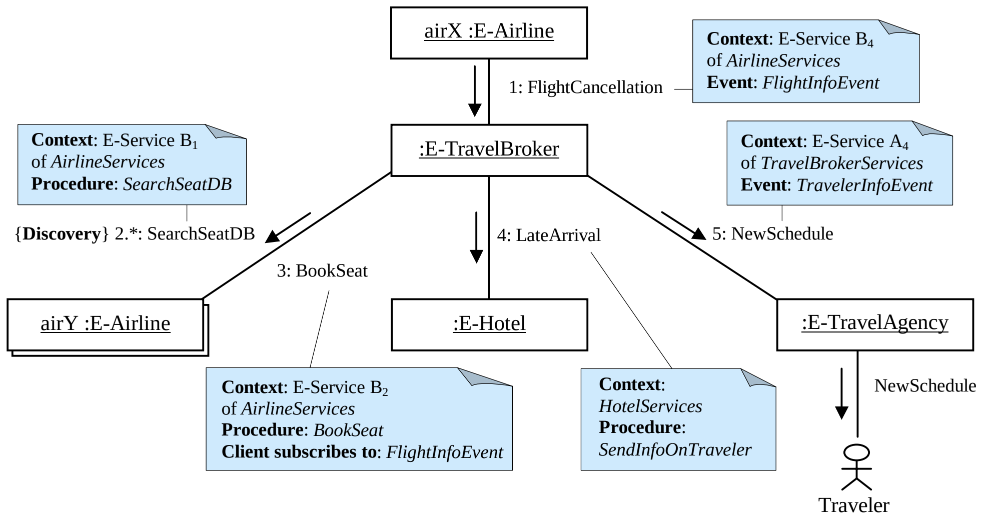
5.3.1 What are the communication styles used to provide and outsource e-services?

This section identifies the communication styles used to provide and outsource e-services. For each e-service, the communication entities (e.g., procedure, event) that realize the e-service are identified by considering UML collaboration diagrams.

The collaboration diagram for the message 'FlightCancellation', sent by E-Airline to E-TravelBroker in case of a flight cancellation, is presented in Table 6. For clarity purposes, it is supplemented by a use case that provides the reader with a textual representation of the collaboration diagram. Both use case and collaboration diagram exhibit the interactions required between e-components to obtain a travel system able to face the "crisis case" shown earlier in Fig. 8.

Table 6 illustrates the decisions about communication styles between e-components. In the world of commerce, relationships are highly dynamic. Travelers may choose freely between hundreds of different travel agents on a journey basis. Likewise, the airlines serving the routes between two cities vary day by day. Despite this change, however, the e-components have to interact correctly. In step one, the E-Airline has to contact the appropriate E-TravelBrokers, i.e., those with delayed customers. In step two, the E-TravelBroker has to locate the airlines serving a particular route during a particular time period. Maintaining these links could be done by a database, but a database solution that would scale would be complex. The collaboration diagram shows that in step one, E-Airline does not need to maintain a database of E-TravelBrokers. Instead, E-Airline broadcasts an event called *FlightInfoEvent* that provides all the subscribers (i.e., the appropriate E-TravelBrokers) with relevant information on flights (e.g., cancellation). Similarly, in step two, the E-TravelBroker uses a discovery process to locate the airlines serving a particular route during a particular time period.

Table 6. Use Case and Collaboration Diagram for the Message ‘FlightCancellation‘

Use Case	FlightCancellation
Description	In case of a flight cancellation, the system modifies the itinerary and informs the traveler about the new itinerary.
E-Components	*E-Airline*, *E-TravelBroker*, *E-Hotel*, *E-TravelAgency*
Assumptions	*E-Airline* knows about any flight cancellation. The traveler has booked a crisis-handling itinerary including flight and hotel. The traveler has allowed the system to modify his travel itinerary without being consulted in case of an emergency.
 airX :E-Airline 1: FlightCancellation Context: E-Service B4 of AirlineServices Event: FlightInfoEvent :E-TravelBroker Context: E-Service B1 of AirlineServices Procedure: SearchSeatDB Context: E-Service A4 of TravelBrokerServices Event: TravelerInfoEvent {Discovery} 2.*: SearchSeatDB 4: LateArrival 5: NewSchedule 3: BookSeat airY :E-Airline :E-Hotel :E-TravelAgency Context: E-Service B2 of AirlineServices Procedure: BookSeat Client subscribes to: FlightInfoEvent Context: HotelServices Procedure: SendInfoOnTraveler NewSchedule Traveler	
Steps	1. *E-Airline* notifies the traveler’s *E-TravelBroker* for the cancelled flight. 2. *E-TravelBroker* searches the seat database of several other *E-Airlines*. 3. Finally, *E-TravelBroker* books a seat going to the nearest airport. 4. *E-TravelBroker* updates the travel itinerary and notifies *E-Hotel* for the traveler’s late arrival. 5. *E-TravelBroker* notifies *E-TravelAgency* of this updated itinerary. *E-TravelAgency* in turn informs the traveler.

The collaboration diagram shows some modeling decisions taken by the architect. For instance:

- The messages SearchSeatDB and BookSeat sent in the context of the e-services B_1 and B_2 of *AirlineServices* (see Table 5) are realized using the procedures *SearchSeatDB* and *BookSeat*. Since these procedures belong to the same service set, they can be grouped in an interface called *AirlineInterface* (see Table 7). In addition, the E-TravelBroker automatically subscribes to the event *FlightInfoEvent* when booking a flight seat.
- The message FlightCancellation sent in the context of the service B_4 of *AirlineServices* (see Table 5) is realized using an event called *FlightInfoEvent* (see Table 9).

For each message of the collaboration diagram, sent by an e-component *A* to an e-component *B*, the architect asks the question: *“What is the appropriate communication style that should be used to send the message from A to B?”*

If the service consumer always uses the same provider, or if there is only a small set of possible providers then **procedure** is probably the most appropriate mechanism on the grounds of efficiency. In this case, the architect adds a note to the message with the procedure name (and possibly signature).

If there are many recipients for the message, and especially if it would be onerous for the service provider to keep track of who should receive the message, then a publish-and-subscribe **event** is probably the best mechanism. Again, the architect adds a note to the message with the event name. In addition, he has to make sure that *B* has already subscribed to this event. A subscription to an event is also shown by using a note related to the message that leads to the subscription.

5.3.2 Interface Specification

When interfaces are identified, the architect should complete an interface specification template (see Table 12 of the Appendix). An interface specification describes the interface in terms of provider (e-component that provides the interface), e-services realized, and goals, and lists the procedures supported by the interface. The *AirlineInterface* specification is given in Table 7.

Table 7. *AirlineInterface* Specification

Interface	*AirlineInterface*
Description	Interface provided by E-Airline that realizes the e-services B_1, B_2, and B_3 of *AirlineServices* (see Table 5). It allows the consumers (e.g., E-TravelBroker) to search the seat database, to book a seat, and to send information about the traveler's ability to execute the schedule (e.g., delay) to the airline system.
Procedures	• *SearchSeatDB*: return list of seats • *BookSeat*: return reference number • *SendInfoOnTraveler*

An interface can be specific to a given application (e.g., *AirlineInterface* is specific to an airline reservation system) or more generic. A generic interface is one in which the interaction is based on the passing of some standard XML document. An example of generic interface is given in Table 8, in which the interface *HotelInterface* supports only one procedure called *SendDocument*. This procedure allows different clients to send documents to E-Hotel specifying requests and/or containing information. The standards for XML documents in e-commerce are being developed by a number of bodies including CommerceOne, Microsoft, and RosettaNet.

Table 8. *HotelInterface* Specification

Interface	*HotelInterface*
Description	Interface provided by E-Hotel that realizes the e-service set *HotelServices*. It allows the consumers (e.g., E-TravelBroker) to send a document specifying a request such as 'Search the Room Database' or 'Book a Room', or containing information related to travelers.
Procedures	• *SendDocument*

Unfortunately, the choice of generic versus specific interfaces is determined, at least in part, by what e-service framework is to be used. In *E-Speak* for instance, the framework layer discovers service providers only by matching attributes specified by the client. The discovery process is independent of service interfaces. Consequently, *E-Speak* interfaces can be quite generic and the system is more flexible, because it is less subject to client recompilation problems resulting from interface modifications.

In *Jini*, the "lookup service" is responsible for finding service providers matching a given interface specified by the client (attributes are used only to distinguish providers supporting the same interface). Since the mach is based on interfaces, generic interfaces make no sense.

In the *Open Agent Architecture*, an interface is expressed as *ICL solvables*. The "Facilitator" uses *ICL solvables* during a matching process (Prolog unification) to dynamically find a strategy to execute a high-level task specified by the client. Consequently, *ICL solvables* must be specific enough to reflect the agent capabilities. Since the task execution is not hard-coded in the client, specific interfaces do not affect the flexibility of the system.

5.3.3 Event Specification

When events are identified using collaboration diagrams, the architect should complete an event specification template (see Table 13 of the Appendix). An event specification describes the event in terms of publishers, subscribers, e-services realized, and goals. For instance, the specification of the event *FlightInfoEvent* is presented in Table 9.

Table 9. *FlightInfoEvent* Specification

Event	*FlightInfoEvent (flightId)*
Description	Event published by E-Airline that realizes the service B_4 of *AirlineServices* (see Table 5). It provides the subscribers (e.g., E-TravelBroker) with information related to the flight *flightId* (e.g., delay or cancellation).
Information	Flight information expressed in a standard document.

Finally, the architect draws an architecture diagram (see Fig. 13) that summarizes the communication styles between e-components. The following UML notations are used:

- A line ending with a white circle expresses that an e-component provides an interface.
- A dependency relationship (rendered as a dotted arrow) expresses that an e-component requires an interface.
- An association (rendered as an arrow) with the stereotype «Event» expresses that an event may be sent via the publish-and-subscribe mechanism.

Fig. 13 shows that E-TravelAgency uses an interface supplied by E-TravelBroker to search and book travel itineraries, and to send information about the traveler's ability to execute the schedule (e.g., feedback on a modified itinerary, delay). In turn, E-TravelBroker uses the interfaces supplied by E-Airline, E-Hotel, E-Transport, and E-CarAgency, to search and book flights, rooms, taxis and cars, and to send information about the traveler's ability to execute the schedule (e.g., delay).

In the other direction, E-Airline, E-Hotel, E-Transport, and E-CarAgency use events to provide E-TravelBroker with any particular information related to their respective activities (e.g., E-Transport uses *TaxiInfoEvent* to inform E-TravelBroker about a taxi driver strike). In turn, E-TravelBroker uses an event to provide E-TravelAgency with any particular information related to a travel itinerary (e.g., new schedule).

In the travel system, events are widely used for providing travelers with crisis-handling itineraries, because they allow e-components to broadcast important information to all the interested parties without establishing explicit connections between e-components, and thus without maintaining a database of interested parties. Indeed, the publishers do not need to know the subscribers. Like advertising-and-discovery, publish-and-subscribe is a powerful mechanism allowing the construction of communities of e-components that collaborate while being loosely coupled.

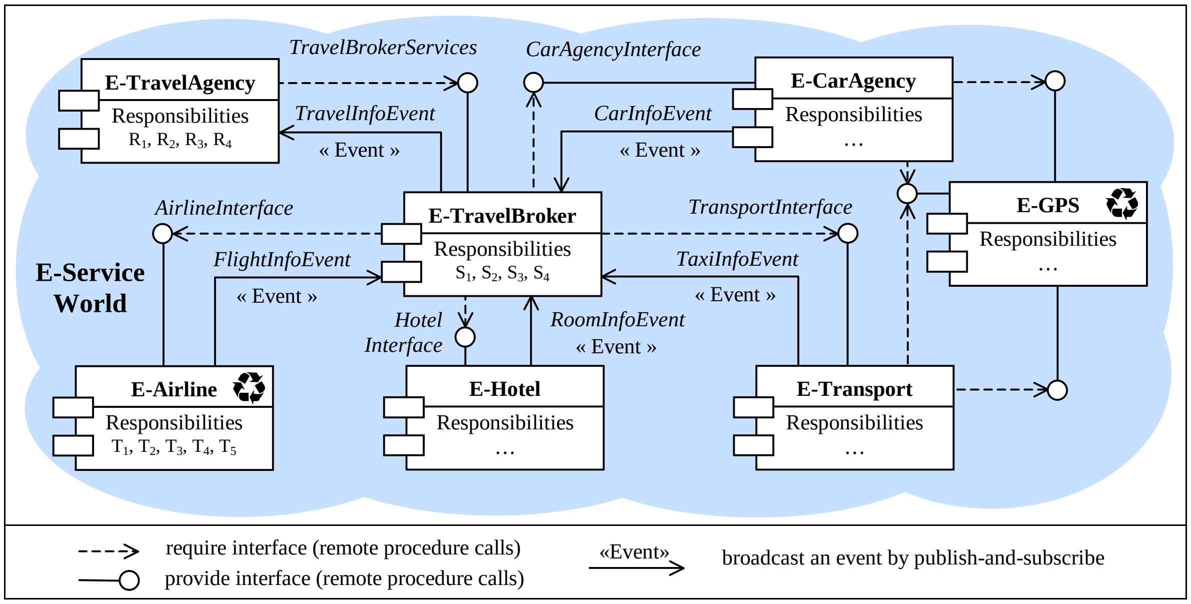


Fig. 13. Architecture Diagram - Communication Level

5.4 Non-Functional Requirements

The next step of the process is to take non-functional requirements such as security, scalability, or fault-tolerance into account.

Security refers to the ability to guarantee the protection of the e-components and their data (data integrity, client and provider authentication, access control, and confidentiality). Ideally, security should be guaranteed by the framework itself. However, since it is not always the case (unlike *E-Speak*, *Jini* and *OAA* do not provide security mechanisms), the architect must introduce some security services. The complexity of these services varies from framework to framework as discussed in Section 5.5.3.

Scalability refers to the ability of the e-service framework to accommodate increasing numbers of e-components. Frameworks like *E-Speak*, *Jini*, or the *Open Agent Architecture* (*OAA*) are scalable in the sense that the number of collaborative *E-Speak* cores, *Jini* lookup services, or *OAA* facilitators can be increased. However, low latency problems may occur; scalability may remain an issue. To overcome these problems, the architect can use some classical load-balancing strategies. For instance, a load-balancing service can be introduced and some providers can be replicated. Then, the load-balancing service selects the providers with respect to their load. This kind of strategy is also appropriated for fault-tolerance purposes.

5.5 Choice of E-Service Frameworks

The final step of the process is to take the semantics of the chosen e-service framework into account. In practice the architect may use this information during the preceding steps of the architecting process. However, we have separated this out into a step by itself to better show the differences between frameworks.

5.5.1 Advertising-and-discovery

The advertising-and-discovery mechanism varies from framework to framework in a way that may affect the modeling process. This section begins with a brief description of the advertising-and-discovery mechanism supported by *E-Speak, Jini*, and the *Open Agent Architecture* (*OAA*).

In *E-Speak*, the "core" is responsible for dynamically finding service providers matching some attributes specified by the client. Consequently, advertising-and-discovery works as follows:

- To advertise an e-service set, the provider supplies a service description to an advertising service. A service description is a document (e.g., XML document) that describes the services by means of attributes that highlight their main characteristics. Attributes are expressed in a vocabulary that should be shared by the community (with the help of standards for instance).
- To discover a service provider, the client must supply a list of attributes. For this purpose, he must know the vocabulary (syntax and semantics) in which the attributes are expressed. In addition, once the service provider is discovered, the client must know how to utilize the services. This may be based on standards, or on contracts established between clients and providers.

In *Jini*, the "lookup service" is responsible for dynamically finding service providers matching a given interface and some attributes specified by the client. Attributes are used only to distinguish providers supporting the same interface. Consequently, advertising-and-discovery works as follows:

- To advertise an e-service set, the provider supplies an interface plus an optional list of attributes to the "lookup service".
- To discover a service provider and to utilize its services, the client requests an interface realizing the services and matching some attributes. For this purpose, he must know the signature of the operations supported by the interface and the syntax of the attributes. In addition, to correctly utilize the services, the client should know the semantics of the operations and attributes.

In the *OAA*, the "Facilitator" is responsible for dynamically finding a strategy to execute a high-level task specified by the client. Advertising-and-discovery works as follows:

- To advertise an e-service set, the provider agent supplies *ICL solvables* to the "Facilitator". A solvable describes a capability offered by the agent. This description may be supplemented by a list of parameters defining the profile of the solvable (e.g., performance, priority, synchronous versus asynchronous). In addition, the provider may publish the vocabulary associated with the solvable by providing a dedicated agent (Natural Language Agent) with verbs and/or nouns with which other agents and users will call the solvable.
- To discover and utilize services, the client must know the corresponding solvables or the associated vocabulary (syntax and semantics).

The descriptions above show that some specific elements, not described in the generic modeling process presented in the previous sections, are required to advertise e-services. These elements are service descriptions including attributes and vocabularies in *E-Speak*, attributes in *Jini*, and profiles and vocabularies in *OAA*. These elements should be identified and specified in the "advertising" section of the e-service set specification (see Table 11 of the Appendix).

Similarly, some specific elements not described in the generic modeling process are required to discover e-services. These elements are attributes and vocabularies in *E-Speak*, attributes in *Jini*, and vocabularies in *OAA*. These elements should be identified and specified in the "discovery" section of the e-service set specification (see Table 10 of the Appendix).

5.5.2 Flexibility

Flexibility refers to the ability of the e-service framework to support e-service modification or suppression. An e-service should be modified (e.g., changes in its interface) without involving the modification and recompilation of its clients in order for them to be able to use the new version of the service. Similarly, an e-service should be removed without involving the modification and recompilation of its clients in order to use another equivalent service.

Thanks to the *OAA* delegation model, *OAA* systems are more flexible than *E-Speak* or *Jini* systems. Indeed, an *OAA* request specifies a high-level task, and then the facilitator is responsible for dynamically finding a strategy to execute the task. In *E-Speak*, the core is only responsible for dynamically finding a service provider that matches some given attributes. The strategy to execute the task is hard-coded in the client. This is similar in *Jini*, in which in addition the service provider must conform to a given interface.

However, as explained in Section 5.3.2, the flexibility of frameworks like *E-Speak* can be increased by using generic interfaces, exchanging XML documents for instance. Given the increasing importance of XML in business transactions, the capability to exchange XML documents is a real asset for a framework.

5.5.3 Mediation

Mediation refers to the ability of the e-service framework to monitor the communications between service providers and clients. Mediation is a powerful mechanism allowing the building of new monitoring services. For instance, it can be used for billing (pay-per-use model) or security purposes. Consequently, the architecture of monitoring services strongly depends on the ability of the framework to mediate the communications.

E-Speak and *OAA* mediate the communications between e-components while *Jini* does not. In *E-Speak* for instance, mediation allows service providers and clients to remain mutually anonymous. Confidentiality is guaranteed thanks to virtual naming. In order to access a service, a client does not use the real address, but a virtual name. The *E-Speak* core keeps a separate name space for each client, and maintains a mapping between real addresses and virtual names. Consequently, confidentiality is free in *E-Speak* that mediates the communications, while confidentiality requires the introduction of a dedicated service in *Jini*.

6 Conclusion

The paper focuses on the issues raised by the modeling of e-service based systems, and presents a method, based on industry standard techniques, dedicated to modeling the architecture of these highly dynamic systems. The main contributions of the paper are the following.

- **Identification of the paradigms encompassed by e-service based systems**
  E-services are delivered by a highly dynamic set of interacting e-components distributed over the Internet. Thanks to mechanisms like advertising-and-discovery combined with communication styles like publish-and-subscribe, the interactions between e-components are flexible, not tightly bounded like the interactions between objects in distributed object-oriented systems. These mechanisms allow the construction of brokers that dynamically combine e-services to achieve high level tasks. In addition, since service clients and providers are not in trusted relationships, mediation and security mechanisms are required.

- **Demonstration of the strength of e-service solutions by means of the travel agency case study**
  The strength of e-service solutions stands in their ability to dynamically react to changes of their environment, in order to provide the *best services* available at a time, and to provide *crisis-handling services* over time. Crisis-handling services may be modified during their execution to take into account changes in the system's state, such as new constraints or new providers offering better solutions.

- **Presentation of three e-service frameworks and their comparison from the modeling point of view**
  Despite technological differences, *E-Speak*, *Jini*, and the *Open Agent Architecture* have a similar vision: the construction of distributed and dynamic communities of e-services that collaborate in order to achieve high-level goals. Their distinctions, mainly in terms of advertising, discovery, flexibility and mediation, must be taken into account during the modeling process.

- **Proposition of a method dedicated to modeling the architecture of e-service based systems**
  The method is illustrated in Fig. 14.

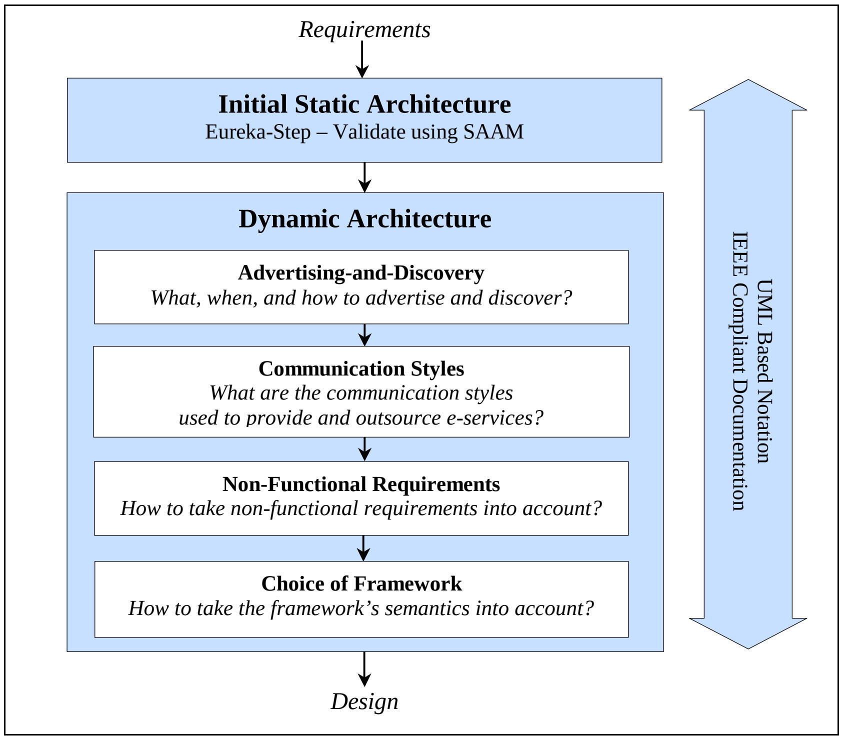

Requirements
Initial Static Architecture
Eureka-Step – Validate using SAAM
Dynamic Architecture
Advertising-and-Discovery
What, when, and how to advertise and discover?
Communication Styles
What are the communication styles used to provide and outsource e-services?
Non-Functional Requirements
How to take non-functional requirements into account?
Choice of Framework
How to take the framework's semantics into account?
UML Based Notation
IEEE Compliant Documentation
Design

Fig. 14. Method for Modeling the Architecture of E-Service Based Systems

The main advantages of the method are the following. First, the method can be integrated with any existing popular object-oriented development method (e.g., Booch, OMT, OOSE, Fusion). Second, the notations are based on UML, which is a visual language widely used and supported by many tools. Third, the documentation of the different models conforms to the IEEE Recommendation for Architectural Description. Finally, since the specificity of e-service based systems stands in their highly dynamic behavior, the method provides the architect with some dedicated techniques for modeling these behaviors.

This method has been successfully applied to several case studies. In particular, it has been used for modeling the architecture of a stockbroker system implemented using *E-Speak*. The application of the method to different examples has demonstrated the pertinence of the approach and its potential to contribute to the quality of future e-service based systems.

7 References

1. H. Fieres, “E-Services in System mySAP.com, A New Perspective in Application Programming,” SAP/HP Alliance, December 2, 1999.
2. P. B. Seybold, “Preparing for E-Services Revolution, Designing your Next-Generation E-Business,” Patricia Seybold Group, April 30, 1999.
3. *E-Speak,* Hewlett-Packard, http://www.e-speak.hp.com/.
4. *Jini Connection Technology,* Sun Microsystems, http://www.sun.com/jini/.
5. *The Open Agent Architecture,* SRI International, http://www.ai.sri.com/~oaa.
6. W. K. Edwards, *Core Jini*: Prentice Hall, June 1999.
7. K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification*: Addison-Wesley, June 1999.
8. *Java Remote Method Invocation (RMI),* Sun Microsystems, http://java.sun.com/products/jdk/rmi/.
9. P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg, “An Open Agent Architecture,” presented at AAAI'94 Spring Symposium, Stanford, March 1994.
10. D. L. Martin, A. J. Cheyer, and D. B. Moran, “Building Distributed Software Systems with the Open Agent Architecture,” presented at the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, Blackpool, Lancashire, UK, March 1998.
11. J. W. Lloyd, *Foundations of Logic Programming*: Springer-Verlag, 1987.
12. *Common Object Request Broker Architecture (CORBA),* Object Management Group (OMG), http://www.omg.org/corba/.
13. *Distributed Component Object Model (DCOM),* Microsoft, http://www.microsoft.com/com/tech/dcom.asp.
14. *Extensible Markup Language (XML),* W3C, http://www.w3.org/XML/.
15. *E-Service Scenarios,* Hewlett-Packard, http://www.hp.com/e-services/strategy/scenarios.html.
16. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice,* Addison-Wesley, 1999.
17. IEEE Draft for Standard, *IEEE P1471/D5.1 Draft Recommended Practice for Architectural Description*, October 1999.
18. M. A. Ogush, D. Coleman, and D. Beringer, “A Template for Documenting Software and Firmware Architectures,” Hewlett-Packard Product Generation Solutions, 2000.
19. *Unified Modeling Language,* Rational, http://www.rational.com/uml/.
20. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*: Addison-Wesley, 1999.
21. D. Coleman, “A Use Case Template,” Hewlett-Packard Product Generation Solutions, 1999.

8 Appendix: Modeling Templates

This section introduces some UML-compatible templates dedicated to the modeling of e-service based systems. The templates can be used for documenting a system's e-components, e-services, interfaces, events, and use cases. They are taken from or inspired by the ones presented in [18, 21]. They do not require the use of any particular formal notation. However, the semantics of any notation must be defined to avoid ambiguities. Templates have two columns. The left-hand column identifies the different fields, and the right-hand column describes the purpose of the fields.

Table 10. E-Component Specification Template

E-Component	A unique identifier for the e-component.
Responsibilities	Describes the purpose of the e-component in terms of its responsibilities.
E-Services offered to Actors	Lists the e-services that the e-component makes available for the actors.
E-Services offered to E-Components	Lists the e-services that the e-component makes available for other e-components. For each e-service, name the communication entity (e.g., interface, event) that realizes the e-service.
E-Services required from E-Components	Lists the e-services that the e-component requires from other e-components.
Constraints	Documents the system-level constraints that the design of the e-component (and its links) must satisfy. These include issues such as • *Multiplicity*: How many instances of this e-component exist in the architecture? Are the instances created and destroyed dynamically? If so, under what conditions does creation and destruction occur? • *Persistency*: Does the e-component or its data need to exist beyond the lifetime of the system? • *Concurrency*: Is the e-component multi-threaded? Does it contain data that needs to be protected against simultaneous reading and writing? • *Component-Actor communication*: What is the nature of the links between the component and actors? • *Out-of-channel communication* (communication not driven by the e-service bus): Are there out-of-channel communications between e-components. If so, what is their nature?
Discovery	Description of the elements (e.g., attributes, interfaces, vocabularies) necessary to discover the e-services required from other e-components.

Table 11. E-Service Set Specification Template

E-Service Set	A unique identifier for the e-service set.
Description	Description of a high level service provided by a set of e-services, in terms of provider (e-component that provides the service) and goals.
E-Services	Name and description of the e-services constituting the set.
Constraints	Any system constraints on the e-service set (e.g., order in which the e-services may be called).
Advertising	Description of the elements (e.g., service description, attributes, vocabularies) necessary to advertise the e-services supplied by the set.

Table 12. Interface Specification Template

Interface	A unique identifier for the interface.
Description	Description of the interface in terms of provider (e-component that provides the interface), e-services realized, and goals.
Procedures	Name and description of the operations (not necessarily atomic) supported by the interface.
Constraints	Any system constraints on the interface (e.g., order in which the procedures may be called, synchronous versus asynchronous).

Table 13. Event Specification Template

Event	A unique identifier for the event.
Description	Description of the event in terms of publishers, subscribers, e-services realized, and goals.
Information	Description of the information associated with the event.
Constraints	Any system constraints related to the event (e.g., event would not flood the network).

Table 14. Use Case Template

Use Case	A unique identifier for the use case.
Description	Goal to be achieved by use case and sources for requirement.
E-Components	List of e-components involved in use case.
Collaboration Diagram	Collaboration diagram that illustrates the use case.
Assumptions	Conditions that must be true for use case to terminate successfully.
Steps	Interactions between actors and system that are necessary to achieve goal.
Variations	Any variations in the steps of a use case.
Non-Functional	List of non-functional requirements that the use case must meet.