Development of E-services

Debbie Lienhart
Hewlett-Packard Company
3404 E. Harmony Road, Fort Collins CO 80528-9599
(970) 898-4227
(970) 898-7734 (FAX)
debbie@fc.hp.com

"An e-service is any asset that you make available via the Net to drive new revenue streams or create new efficiencies." That's the official HP definition of an e-service. In this paper we'll examine some of the implications for developers.

The first question to be answered is "What is an asset?" An ***asset*** is something of value to someone. Assets are tangible things, information, or even access to people.

For example, at HP I can see my personnel records from my desktop browser. If I need a copy of my evaluation, I can access the database and print the information I need myself; saving the time and expense of a person answering my phone call, printing my evaluation, and sending it to me. This is a case where an e-service is creating efficiency for HP by saving both my time and the personnel department's time.

Developing an e-service that uses data already in electronic form is one thing, but how do you make tangible assets, like people and things, available over the net? Some things are easier to provide than others. For example, when you buy a tambourine over the Internet, a bunch of data is moved around, and FedEx delivers the new instrument to your front porch as a programming side effect.

It takes more creativity to make assets that aren't being purchased available over the Internet. How might you make something like a shuttle bus available over the Internet? HP Lab's CoolTown includes a web appliance in the form of a real shuttle bus, with embedded microprocessor, a global positioning system (GPS) and a wireless network. An e-service tracks the bus, making the location information available for display or use by other e-services. Or how about making the skills and knowledge of a person available as an e-service? As our user feedback and questions increase, we'll be looking for new ways to provide information promptly and courteously to as many of our users as possible.

Some Interesting Programming Attributes of E-services

Now that we understand some e-service basics, we'll explore some of the interesting programming attributes of e-services. We'll use the following example of a video advisor e-service to illustrate many of these attributes.

One of my interests is traditional fiddling. I've played music most of my life, developed an interest in traditional music about ten years ago, and started learning to play the fiddle about six years ago. One of the challenges of learning to play traditional fiddle music is that there are a lot more people who want to learn than people who can teach, and the teachers and learners don't

necessarily live in the same area. One way to learn traditional music is to go to music camp - a week of intense music instruction and fun - and then spend the next 51 weeks of the year trying to remember everything you learned. How does this relate to e-services?

Let's say I wanted to provide a reference for people attending a music camp. It should be possible to put enough information on a computer CD that a music camper could refer to it during the year to remember what was taught. The CD could contain a web site with web pages and audio and video examples. I know how to create a web site and put it on a CD, and even how to edit audio, but I haven't edited video before.

Where the e-service fits in

To create the video portions of the CD, I need to have video capture, editing, and compression capabilities; which involves purchasing a video camera, one or two cards for my computer (I'll need to do a little more research to know how many cards are needed), video editing software, and perhaps video formatting software. I may even need to upgrade my PC to make this all work. Right now I'd pay money for an e-service that could advise me on which set of components would work best together to meet my needs, and then identify the least expensive places to order them with reasonable delivery times. Such an e-service would save me time and reduce the risk that I might buy the wrong components.

Let's look at what will need to happen for my dream e-service to come true:

The e-service would need to have a browser interface to interact with me. The browser interface would ask a bunch of questions to determine my technical ability, background, frustration tolerance, priorities, and preferences. (http://www.activebuyersguide.com is one example of this kind of e-service.) Behind the scenes, it would query various sources to find information about the potential components, including general costs, product reviews, usability attributes, and performance characteristics. It should be able to find and process information on how well the various items work together. After the e-service presents reasonable alternatives and helps me narrow the options, it would then find all of the suppliers of the pieces, present them based on my preferences, send orders to the selected vendors, track the fulfillment process, and keep me informed with e-mail as the status changes.

It starts with programmatic access

When most people think about Internet applications, they think of interacting with web pages in their browser. This is similar to having an application with a graphical user interface. It's great if your users are all accessing the application from the GUI, but is a royal pain if the application needs to be accessed from another application. My video advisory e-service needs to query the necessary data from various sources. For the data sources to participate in the virtual corporation formed to solve my problem, they will need to provide programmatic Internet access to their assets (data, in this case).

But how can they communicate this data?

E-services use vocabularies, formatted as XML schemas, to share data

E-services use files formatted in Extensible Markup Language (XML) to contain the data they are exchanging. XML is both computer-readable and human-readable. XML can be parsed by a computer, which means it can be validated and useful tools can be created. With style sheets to control presentation and appearance, people can read XML documents in a web browser.

To communicate information, the sender and receiver must share a language. With XML, the language is described using a schema, with many object-oriented attributes. For example, if the sender and receiver are discussing shoes, one may understand a specialization, like "tennis shoes," while the other does not. If the sender sends information more specialized than the receiver needs, the receiver will know enough about the schema to throw away the more specialized information. If the receiver needs the specialized information and the sender doesn't provide it, the receiver can either decide to gracefully handle the more general case (perhaps by using only the general information about shoes) or it can choose not to work with that sender.

Many industry groups have started to define the vocabulary for their domain. For example, HP is involved in RosettaNet, an organization that is defining the electronic business interfaces needed for Electronic Components (EC) and Information Technology (IT) supply chains. As various industries have a need to conduct business-to-business electronic commerce, they will work together to define the vocabularies they need.

E-services use component architectures

One of the concepts behind e-services is that a temporary ***virtual corporation*** is created to handle each transaction. In my video advisor e-service, the virtual corporation that handles my case will likely be different than that assembled to handle the case of a professional video editor, and it will likely be different than that assembled to handle my request next month when new products and services are available.

To gain the business that can be generated by participating in virtual corporations, e-services must be constructed for inclusion in a dynamically composed component architecture. Thus, all but the very leaf e-services must have component architectures.

For example, let's say that an on-line magazine provides product reviews for its subscribers for a fee. To participate in virtual corporations it provides programmatic access to the information, for a fee, to my video advisor e-service. Because one of the magazine's core competencies is reviewing products, but not processing on-line fees, it may use a billing e-service to collect the fee. Thus, the review provider e-service is both constructed from multiple components (the billing e-service, the database of product reviews, and perhaps others) and is a component in the video advisor virtual corporation. The advantage of using components in a tightly coupled application, like the magazine's use of the billing e-service, is that is can take advantage of capabilities and efficiencies outside the company's core competency. The advantage of a component architecture in a loosely coupled service, like the video advisor, is it provides flexibility for assembling the set of e-services that form the best virtual corporation to meet my needs.

Evolution of Applications To General E-service Computing

We can view the path from web-unaware applications to general e-services computing as a series of plateaus. These are browser-enabled applications, internet-architected applications, service-delivering applications and dynamically composed services. In this section we look at the characteristics of each plateau, as well as the requirements for advancing to the next plateau.

It is likely that many application or service providers can meet their business needs on one of the plateaus short of the grand e-service computing vision, illustrated by the video advisor example. Since each plateau has increasingly difficult requirements and uses less mature technology, it is prudent to consider where your application really needs to be and not aim for a higher than necessary plateau. There are many projects that have failed because they took on more extensive requirements than necessary – it's not just a characteristic of e-services development.

Browser-enabled applications

With a browser-enabled application, the user has access to the application from their web browser. Some examples:

- **Enterprise**: Employees gain access to business applications (including employee phone list, submitting expense reimbursement requests and conference room reservations) over the intranet through their web browser.
- **Business to business**: Businesses can share data or provide access to other assets over the internet or with an extranet. For example, if two companies are collaborating on a project, they can create a database with access via a protected web site to interact with the data in the database.
- **Business to consumer**: A business can provide access for their customers to place or check on orders.

Application description

Browser-enabled applications generally have two major parts, the user interface and the main application. The user interface is constructed from one or more web pages that are viewable with a web browser. The main application looks like any non-web-enabled application. For example, it may contain a database with some processing code, it may be written in any computer language, it may have a very complex architecture – it really doesn't matter.

For the main application to operate in this role, it must not require any input, or be able to accept input from another computer program. For example, command line interfaces, a message interface and input file files are all ways an application can accept input. The main application must also be able to output any information in a way that can be read by another program, or is directly viewable with a web browser. For example, if the main application produces a structured output file, then a script can be written to format it into HTML for display in the browser. Alternatively, the main application could provide its output in HTML, though that usually is a bad idea architecturally because too many user interface details get hard-coded in the main application.

The implementation of the interface between the user interface and the main application will depend on the interface to the main application. Many times these can be implemented with the user interface gathering input via an HTML form, which is processed by a cgi script. The cgi script then executes the main application, giving it parameters based on the user input, and then formats the results from the main application, displaying them as a dynamic HTML page.

Software development implications

The user interface is developed as a web design project. The web pages forming the user interface are written in HTML. The cgi programs are usually written in a scripting language, such as perl or python, but can also be written in C or other compiled languages, especially if execution speed is critical. If the user interface is complex, then testing becomes a concern. Differences in browser behavior need to be designed for and tested.

There are no special development implications for the main application.

Web-architected applications

With web-architected applications, the application is implemented using internet technologies. Internet technologies allow the development of more dynamic, scalable and portable applications, by allowing the developer to focus more of their effort on business logic, and less on application infrastructure.

The user will not see a functional difference between a browser-enabled and web-architected application.

Application description

Web-architected applications are generally N-tier applications, including databases, web servers, application servers and clients. Additional internet technologies may be used to implement features for security, availability and management.

Software development implications

It is often easier to manage the development of web-architected applications with an iterative-based lifecycle, rather than a waterfall lifecycle. The main reason is that web-architected applications are more like continuously running organisms than traditional computer programs that execute once and stop. This dynamic behavior can be difficult to predict, especially when you are using new or less mature technology. Using an iterative-based lifecycle lets you learn how the system will operate early in the project, reducing project risk.

Service-Delivering Applications

With service-delivering applications, software products are surrounded by e-services. The services add value to the products.

For example, I might add a service to my browser-enabled or web-architected application that supports varying pricing models for the service based on the time of day. I can provide an option

for the user to submit a job anytime during the day and guarantee that it will be done when they get in the next morning, and be charged the less expensive overnight rate. As an application, or service, provider, I can minimize my capitol equipment expense by balancing the workload through the night, and the user gets the benefit of the cheapest timeslot without having to submit the job from home in the middle of the night.

Application description

Service-delivering applications have a combination of applications and services. Architecturally, the services look like applications. Some service components may be out-sourced. For example, you may want to incorporate a shipping service from a shipping company, or a billing service from a billing company. The multiple applications will need to communicate with each other, but this communication can use lightweight, proprietary mechanisms.

Software development implications

There is a relatively lower barrier to entry for services, so companies providing them will need to be able to evolve them quickly – at internet speed. This will require very low overhead software lifecycles and development processes.

Development organizations may need to manage different parts of the service-delivering applications separately, perhaps using iterative-based lifecycles on the rapidly evolving components and more traditional lifecycles on the more stable components.

There may be additional development requirements to deliver the service components. For example, if part of the service includes quality of service guarantees, then the system will need to incorporate hooks and components to implement this service.

Dynamically Composed Services

Dynamically composed services are composed of e-service components, which assemble themselves into a virtual corporation to service a user request. The video advisor service from the first part of the paper is a dynamically composed service. The e-service components may come from a variety of companies and it is likely that a different set of components would be assembled for each user request.

Dynamically composed services are the ultimate in e-service computing because they can exploit the flexibility of the web to benefit both the service provider and the service consumer. This flexibility is a result of the combination of programmability and composability of e-service components.

By participating in dynamically composed services, e-service component providers, such as the information sources and billing components in the video advisor service, gain additional channels and business for their service. Because there will be room for very niche services, there will be low barriers to entry for new service providers.

The provider of the top-level service, the video advisor in this case, can deliver a more personalized service. As the service becomes increasingly personalized, it can be priced as a

consulting service, rather than an information delivery service, resulting in higher profits.

E-service components share some requirements, such scalability and high availability, with enterprise components, but must also require higher levels of autonomy and goal-directed behavior. They must be able to complete their mission, even if their dependent components fail. This requires tracking their progress and knowing when to try alternate paths.

Since e-service components participate in many different dynamically composed services, they need to be able to operate in different ways. For example, in one dynamically composed service, they may perform a single transaction, while in others they may participate in a complex work flow for an extended period of time. Depending on the architecture of the base application underlying the service, providing this flexibility could be an implementation challenge.

E-service components cannot assume that they are operating in an environment of trust. Since the services are dynamically composed, each component has to consider the possibility that one or more of the other components are rogues. For example, when an e-service looks for a business recommendation, how can it be sure it got advice from the Better Business Bureau and not the Bad Business Bureau? Solving these trust issues will be a critical factor in general adoption of dynamically composed services. There is a significant amount of research being done in HP Labs in this area.

Application description

Dynamically composed services look like a bunch of objects of varying sizes, complexity and construction. They share some traits – the ability to exchange information through message interfaces using XML documents, and the ability to communicate via the internet.

Most objects will have some messaging interface code, an XML parser and the code to support the service they provide. For example, a service might need to access a database or other application software.

The applications themselves are non-deterministic. Not only are they message-driven, but a different set of components could be assembled each time the service is provided.

Software development implications

Successful dynamically composed services will be developed with very close ties between the business planner and software developers. There will need to be tools for both business planners and software developers, and these tools will need to share data between them.

E-service components participate in non-deterministic systems. Since non-deterministic systems are very difficult to test, it will be critical that the initial code be free of coding defects. Tools that generate major portions of the code, along with disciplined development practices will help ensure the robustness of the e-service components.

Conclusions

It is helpful to think about the programming attributes of e-services in visualizing the software development requirements. E-service development builds upon programming concepts such as

programmatic access to applications, message interfaces and component architectures that have been used for many years.

E-services allow companies to deliver higher value, more customized services to their customers. The general e-services computing model presents significant challenges to developers and development organizations because of the complexity of developing and deploying autonomous, goal-directed components that don't operate in an environment of trust, with an internet speed software development process.

It is useful to think about the range of e-service possibilities between web-unaware applications and the general e-service computing model as a series of plateaus. Web-enabled applications, internet-architected applications and service-delivering applications all allow organizations to benefit from delivery of services over the internet, or intranet, while building the experience needed to develop dynamically composed services.

HP Developer's Resource (http://devresource.hp.com) contains information and tools to help developers and development organizations adopt internet technologies and e-services computing.