DynaDesigner: A Tool for Rapid Design and Deployment of Device-Independent Interactive Services

Loren Terveen, Elena Papavero, Mark Tuomenoksa

AT&T Bell Laboratories
{terveen, elenap, mlt}@research.att.com

ABSTRACT
DynaDesigner is a tool for creating, testing, and deploying interactive services to be delivered on devices such as telephones, TVs, and PCs. A key feature is that it supports *device-independent* service design – a service is designed once, independent of any particular device. This eases the design and maintenance task for service providers and makes services easier for consumers to use, since they are consistent across devices. DynaDesigner has been used to design and deploy many services. With DynaDesigner, services can be designed and deployed in hours.

KEYWORDS: service creation tools, device-independent design, end user programming, consumer systems

HELP SERVICE USERS BY HELPING SERVICE CREATORS

The convergence of computers, communications, and consumer electronics brings a new urgency to the problem of creating easy to use systems – now all people with telephones and TVs, not just PC owners, are potential users. Further, the proliferation of devices raises the challenge of creating *device-independent* services – consumers may want to use a single service such as home banking on different devices, such as telephones, TVs, PCs, or PDAs, and they will want consistency across devices. We have responded to these challenges by creating a service authoring tool called *DynaDesigner.* Direct manipulation, form-based editing, and graphical simulation are used to design and test services. Code is generated that implements the service logic on standard hosting platforms. Device-specific interpreters enable the same service to be delivered on many devices and guarantee a consistent, high quality interface for each device.

TRANSACTIONAL SERVICES

DynaDesigner targets the class of transactional services that includes activities such as home banking, bill paying, ticket purchasing, and catalog shopping. In such services, a consumer completes a transaction by reading or listening to information, selecting options from menus, and supplying information such as personal identification numbers.

Such services typically are implemented in a distributed, network-based architecture. The program that implements the service runs on a *hosting platform* – a computer on the service provider's premises or in the telephone network equipped with special purpose voice and data transmission hardware. The service provider's databases typically are on a large mainframe computer. And the service is delivered on a consumer device, such as a telephone, TV, or PC. The hosting platform, database machines, and delivery devices are connected by a combination of telephone lines and high speed data links.

DynaDesigner

DynaDesigner is a service creation tool that addresses the following goals:

- *Service creation should be fast and easy.* Competitive pressures make speed necessary; speed requires an easy to use system.
- *Services created should be of high-quality.* Making service creation fast and easy is of no use unless consumers want to use the services.
- *Services should be device-independent.* A device-independent representation of services aids service users by ensuring service consistency across devices and aids service providers since they have just one service to design and maintain.

DynaDesigner achieves these goals with two major features:

- Service authoring consists of specifying the service logic and information content, rather than designing an interface. DynaDesigner uses interface "templates" to generate an appropriate interface for each delivery device from the same logic and information content. This helps service providers by making service design and modification faster and easier. It helps end users by guaranteeing a consistent, high-quality interface across services and devices. And it is the key enabler of device independent services.
- DynaDesigner supports end-to-end service creation. All aspects of the service, including logic, interaction, data access, and call transfer can be specified, and code to implement the service on a standard hosting platform is generated automatically. These features are required to make service deployment and modification fast.

Our service creation building blocks include *computational* (or *logic*) objects, such as objects for accessing databases, branching based on data comparisons or on the time, date, or day of week, and transferring calls to customer service representatives. Clearly, the logic of a service is device-independent – the challenge is coming up with a device-independent way of specifying user interactions. We do this by providing a set of generic *interaction* (or *dialogue*) objects that encapsulate common types of interactions – such as *Present Information*, *Get Input*, and *Present Options* – but do not specify the surface details of how the interaction should be carried out. Each type of object has one or more editors for filling in the information required for that object.

A designer specifies the logic of a service by selecting objects from a palette, positioning them on a design pad, then linking the objects. For example, a home banking application might begin with a *Present Information* object that welcomes the user, continue with a *Get Input* object that asks the user to enter a personal identification number (PIN) and verifies the PIN is valid, and next contain a *Present Options* object that offers the user a set of home-banking services.

While we keep the core of the service specification device-independent, we also provide principled ways to specify and use device-specific information. First, a device-specific interface template for each interaction object is used to generate the speech or display and manage user input. For example, suppose a *Present Options* object contains three options, say "Account Balance", "Transfer of Funds", and "Order New Checks". The template for spoken menus and text-to-speech synthesis would generate the spoken instructions "For Account Balance, press 1", "For Transfer...". Users could select an option by pressing the appropriate key on the touchtone pad. The template for TV display screens might result in each item being presented on a separate, numbered line. Users could select an option by pressing the appropriate button on a universal remote control. Second, device-specific editors collect information that enhances the presentation of a service on a particular device. For example, a phrase manager for spoken output allows text to be broken up into phrases, each of which can be spoken using a professionally recorded voice, and an appearance editor for screen output allows designers to specify bitmaps to use as screen backgrounds and tailor display properties such as font, background color, etc.

At any time, a service designer may simulate a service, to determine whether the logic is correct and get a feel for what the interaction would be like on a specific device. When the service is complete, DynaDesigner automatically generates the code to implement it on a standard service hosting platform. Currently, code is generated for the Conversant, a special purpose AT&T computer that resides in the AT&T network or on the premises of a service provider. It can send and receive voice or data over telephone lines and can communicate with other computers as necessary, for example, to access a database.

We have developed DynaDesigner in close collaboration with service providers, building many services to ensure ease of use and broad coverage. DynaDesigner went into production use in August 1994. More than 20 voice services have been deployed. Using DynaDesigner dramatically decreased the time it took to design and deploy a service. The application of DynaDesigner to interactive TV services is newer, and it was this that led us to develop the device-independent interaction objects. To date, we have designed several services based on these new objects and will be implementing and deploying these and other such services in the near future.

RELATED WORK

DynaDesigner is related to cross-platform GUI construction tools and end user programming environments. Like a GUI builder, it enables the design of an interactive system that can run on different devices. However, with DynaDesigner, designers specify dialogue structure and content, not widgets and layout. And rather than delivery across different windowing systems and interface toolkits, DynaDesigner services are generic across a range of devices, from televisions to telephones. As an end user programming system, DynaDesigner enables users who are not programmers to create applications. Unlike general purpose visual programming [1] or state machine [3] systems, DynaDesigner is a domain-specific system. It provides a set of high-level, domain-specific abstractions for building applications. Other systems exist for creating voice dialogues, both commercial tools such as Visual Voice and TFLX, and research prototypes such as the Voice Dialog Design Environment [2]. Our work is distinguished by our provision of high-level, device-independent building blocks. Further, we automatically generate the code to implement services as network-based, distributed applications, which typically means that many more customers can be served.

CONCLUSIONS

We conclude by summarizing the lessons this work has taught us. First, by providing high-level abstractions for creating services, we both ease the task of service authoring and enable device-independent services. Service authoring is neither a programming task nor an interface design task, which leaves service authors free to concentrate on the service content. (We are continuing to develop our building blocks in response to user feedback; for example, we have created objects for dealing with databases that are simpler to use, encapsulate both data access and user interaction, and are more cleanly device-independent.) Second, end-to-end service authoring requires addressing infrastructure issues; thus, we generate code to implement services on standard hosting platforms, to handle distribution of computation in network-based services, and to interface to legacy databases. The combination of high-level building blocks and end-to-end service generation makes DynaDesigner a very usable and useful tool.

REFERENCES

1. Glinert, E.P. Towards "Second Generation" Interactive, Graphical Programming Environments, IEEE Comp. Society Workshop on Visual Languages, 1986, IEEE Press, pp. 61-70.
2. Repenning, A. and Sumner, T. Using Agentsheets to Create a Voice Dialog Design Environment, In Proc. ACM/SIGAPP Symposium on Applied Computing, 1989, ACM Press, pp. 1199-1207.
3. Wellner, P.D. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation, In Proc. CHI'89, 1989, ACM Press, pp. 177-182.