Product Definition Through the Design-by-Use Process

Steven L. Greenspan
John J. Wisowaty
Raymond E. Bright, Jr.

To promote product quality, a design process should represent the needs of all who will use, develop, or modify the product. The service creation environment (SCE) for the A-I-Net™ adjuncts, a family of products that allow service providers to control the definition, development, and evolution of advanced intelligent network services, was developed with a design-by-use process. This process is an iterative cycle of analysis, prototype construction, and user evaluation. It uses human factors and system prototyping techniques to involve potential users of the product early in the product definition and design stages.

Introduction

To promote product quality, the design and evaluation of products and services should be shaped by the demands and organization of the work environment in which they will be used. This is a central tenet of the ***participatory design*** practices successfully applied by AT&T, Digital Equipment Corporation (DEC), International Business Machines (IBM), XEROX PARC, and the ESPRIT/HUFIT European project.[1–4]

To encourage the acceptance and integration of a product into the work environment, participatory design practices maintain the following guidelines:[3–5]

- The goal of participatory design practices is to improve the quality of work life.
- The design decision process should include representatives of those who will use the product or will be affected by the design process. Collectively, these participants should represent diverse sources of expertise.
- The process should be iterative, with repeated evaluations by the user community as the product is being defined and designed.

Participatory design practices generate timely, usable information that is directly applicable to critical design decisions.[1] DEC encouraged their employees to participate as a user community throughout the design and development of their EVE text editor. Similarly, when XEROX PARC was designing *Trillium*, a computer-based software tool for designing user interfaces for copiers, its employees were designated as the user community.

When participatory design practices are used extensively to design new products, there is typically an existing community of users experienced in the task to which the new product will be applied (e.g., editing text or designing user interfaces for copiers). In contrast, the design-by-use practices described in this paper were used in the design of a new product for which no experienced user community existed.

The Product Domain

To meet rapidly evolving market demands and requests for customized service, service providers (e.g., the local telephone companies in the United States and throughout the world) want an advanced intelligent network, a network of switching offices and adjuncts that enables them to define, design, develop, test, manage, and maintain new services. The logic for these advanced intelligent network services will reside on network adjuncts, such as the service control point (SCP). (See Panel 1 for definitions of abbreviations and acronyms.) Traditionally, equipment vendors, including AT&T, were responsible for service design and development. With the advent of the advanced intelligent network, service providers will assume these responsibilities.

Panel 1. Abbreviations and Acronyms

OAM&P	operations, administration, maintenance, and provisioning
SCE	service creation environment
SCN	service circuit node
SCP	service control point
SLL	service logic language
SMS	service management system

The speed and reliability with which advanced intelligent network services can be developed by a service provider is a key factor in determining whether a service provider will accept an equipment vendor's advanced intelligent network products.[6,7] To help manage and simplify service creation in the advanced intelligent network, AT&T has developed the A-I-Net service creation environment for its A-I-Net family of advanced intelligent network products. The service creation environment enables service providers to create the service logic and data used on the A-I-Net service circuit node (SCN) and service control point, and managed by the service management system (SMS).

The service control point contains a real-time database system that executes subscriber-specific logic (e.g., a business customer's call-routing plan for an 800 number). The switching office queries the service control point, which returns instructions to the switching office on how to continue processing the call. The service control point can screen area code 900 and 976 calls, selectively blocking some calls and allowing override permission with an authorization code. It can also screen incoming calls and route them based on time and day, and the number and location of the caller.

The service circuit node supports services that use service circuits such as facsimile, voice, text-to-speech, speech recognition, conferencing, and tone and call progress detection/generation devices to calling and/or called parties. The service circuit node can also initiate scheduled "wake up," "remind me," and message broadcast calls. The service management system allows service administrators and their service subscribers to create and maintain subscriber data (e.g., customized routing plans that are used by service control point and service circuit node services). Figure 1 shows the logical relationship among these products.

To define and design the service creation environment, the service creation environment design and development team used a version of participatory design practices that we call design-by-use, which is described briefly in Reference 8.

The Design-By-Use Process

The design-by-use process is an iterative cycle (see Figure 2) in which the user community, designers, and developers:

- *Clarify the design and capture expert knowledge*—For A-I-Net service creation environment design and development, the design and development team talks with service creation environment designers and developers and subject-matter experts from related fields (service design, switching software and hardware design, call processing, user-interface design, and Operations, Administration, Maintenance, and Provisioning [OAM&P]). The design and development team clarifies product goals, formulates design principles, and revises the service creation environment design.
- *Design and develop the prototype*—The design and development team designs low-fidelity (sometimes paper) and high-fidelity (working software and hardware) prototypes.
- *Observe potential users working with the prototype to accomplish realistic tasks*—Service creation environment users design, code, and test selected subsets of probable A-I-Net services during a period of concentrated work lasting from several days to several weeks. The design and development team then analyzes and discusses the observations with users, and the results are incorporated in later iterations of the cycle. This leads to the definition and design of more fully capable prototypes. Moreover, because the process requires training users, each cycle of the process produces more complete training materials and a more sophisticated user community.

Before we describe what we did and learned at each stage of the design-by-use process, we discuss one alternative process. Traditionally, many products are defined through a series of conversations with systems engineers, product management, the development community, and customers. These conversations are distilled into product requirements, which are then given to a team of developers. Although such conversations are

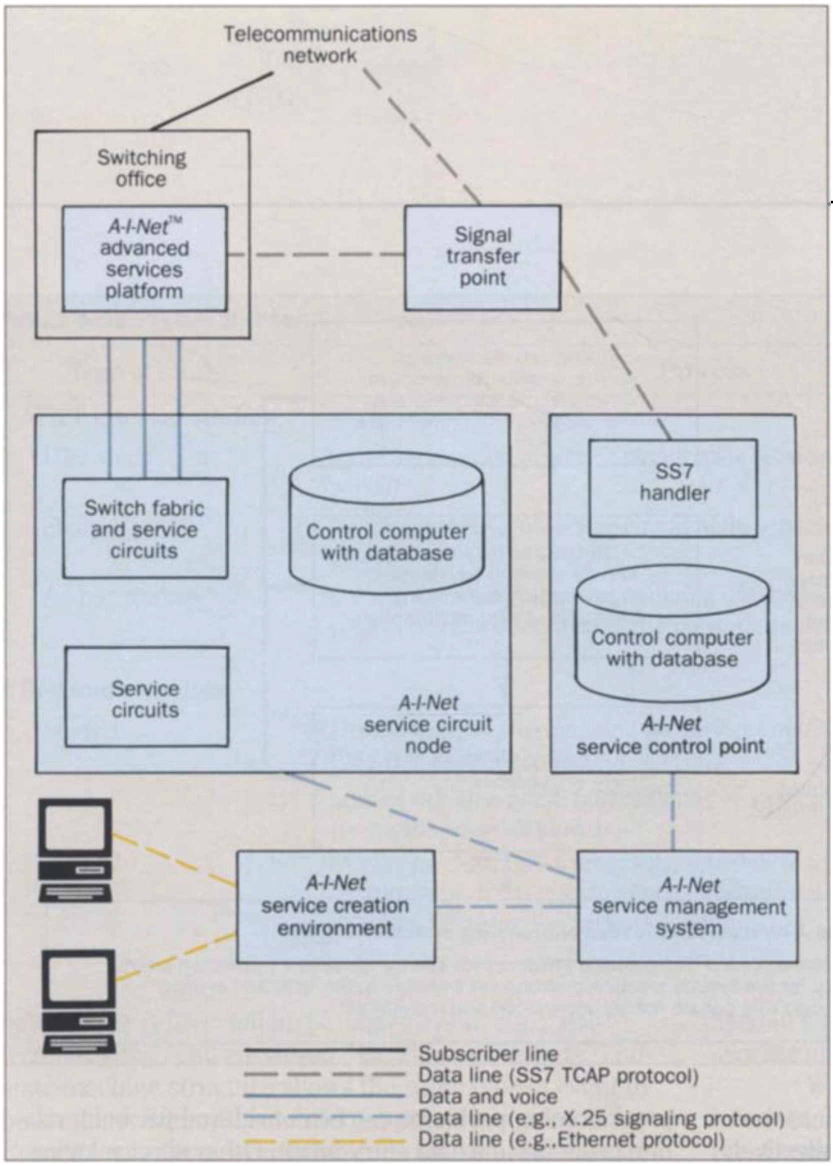

Telecommunications network
Switching office
A-I-Net™ advanced services platform
Signal transfer point
Switch fabric and service circuits
Control computer with database
Service circuits
A-I-Net service circuit node
SS7 handler
Control computer with database
A-I-Net service control point
A-I-Net service creation environment
A-I-Net service management system
Subscriber line
Data line (SS7 TCAP protocol)
Data and voice
Data line (e.g., X.25 signaling protocol)
Data line (e.g.,Ethernet protocol)

Figure 1. The family of A-I-Net products is shown in color. The advanced services platform, signal transfer point, and service control point all rely on SS7 TCAP protocol, which is an applications-layer signaling protocol used between network elements. The service circuit node receives calls from trunks and lines, while the service management system and the service creation environment communicate with the service circuit node and service control point through data connections.

very important (and comprise the first phase of the design-by-use process), this analytic, nonempirical approach works best with products that are already in use and undergoing modification.

The A-I-Net service creation environment had no existing community of advanced intelligent network service developers or any extant product that could be used to create advanced intelligent network services. The design-by-use process created a small user community in which the design and development team could iteratively observe and evaluate our service creation concepts in action. It provided user feedback early in the product definition and design process. By prototyping and evaluating the service creation environment, the design and development team gained valuable insights into user needs and changing product requirements.

The importance of protyping and user evaluation is evidenced by the following quotations from one research and development community[2]: "We didn't anticipate this," "Why is user testing so late?" and "It's not broken; that's how it's supposed to work." Involving users early in the design process and testing prototypes with realistic work assignments enables designers and potential users to understand user needs and product behavior long before the product is released.

Preliminary Design Clarification

Before 1988, there was no existing user community or commercially available product on which to base our designs. Therefore, we began by scrutinizing available documentation on customer needs and service creation concepts,[9,10] and we interviewed expert software developers who had contributed to AT&T switching products. Typically, these experts had 10 to 20 years of experience developing or maintaining services, had worked on various phases of the software life cycle (e.g., design, coding, testing, or field support), and had been involved in more than one area of telecommunications

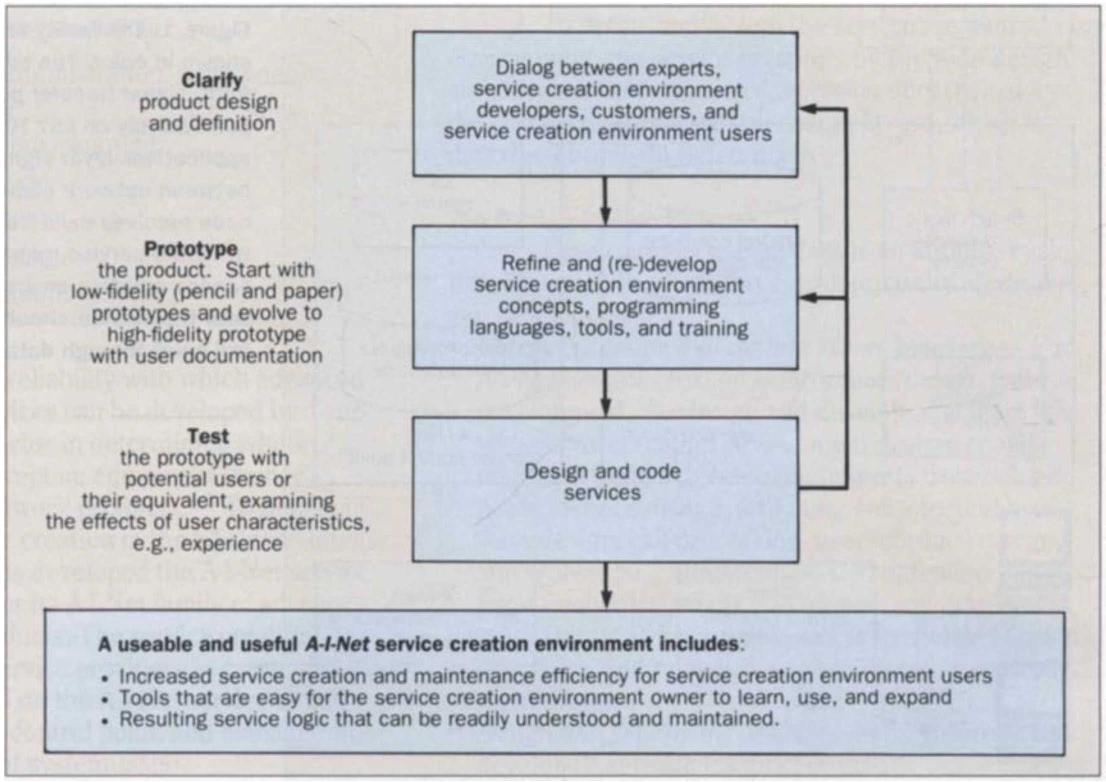

Clarify
product design
and definition
Dialog between experts, service creation environment developers, customers, and service creation environment users
Prototype
the product. Start with low-fidelity (pencil and paper) prototypes and evolve to high-fidelity prototype with user documentation
Refine and (re-)develop service creation environment concepts, programming languages, tools, and training
Test
the prototype with potential users or their equivalent, examining the effects of user characteristics, e.g., experience
Design and code services
A useable and useful *A-I-Net* service creation environment includes:
• Increased service creation and maintenance efficiency for service creation environment users
• Tools that are easy for the service creation environment owner to learn, use, and expand
• Resulting service logic that can be readily understood and maintained.

Figure 2. The design-by-use process is iterative, integrating the judgment of experts and users with empirical evaluations of those judgments.

for more than one type of switching system. Others were experts in language design or human factors. Collectively, these experts helped the design and development team define a conceptual foundation for service creation environment design and, later, helped interpret the results of user evaluations of our service creation environment prototypes. To ensure that our changing view of service creation was consistent with that of the larger technical community, we participated in standards forums about the intelligent network.

As a result of our preliminary discussions on product definition, we decided to develop application-oriented languages for service creation. Application-oriented languages are specialized computer programming languages designed to handle well-specified tasks using a particular conceptualization of the task. For example, Lotus-1-2-3® software uses the concept of a spreadsheet to handle many types of data management problems. (Lotus 1-2-3 is a registered trademark of the Lotus Development Corporation.)

Service creation and management tasks can be conceptualized in a variety of ways. For example, some programming problems can be considered parameterization tasks (as in a data entry form). Other service logic programming tasks are well represented by spreadsheets or decision flows. For instance, call-routing logic can be represented as a branching set of sequential decisions terminating in a directory number.[8] These conceptual models have been used extensively in telecommunications service management.

However, many advanced intelligent network service descriptions require fine control over asynchronous, multiple-network resources. During a single call, these services may have to control parallel requests for network resources (e.g., concurrently querying a database and interacting with a calling and/or called party). After discussing these needs with our subject-matter experts, we concluded that the logic for these services could be well represented as a finite-state machine. Finite-state machines traditionally have been used to diagram telecommunication processes and service flows.[11] Within this model, A-I-Net service programmers write software as a set of *event handlers*, where each event handler describes responses to an end user,

Table I. Design-by-Use Studies

Type of study	Process
AT&T internal studies	
Pilot study	SLL programming of service circuit node services (using paper and pencil)
Study 1	SLL programming of service circuit node services (using paper and pencil)
Ad hoc studies	SLL and decision-graph programming of service circuit node and service control point services (using the network model)
BellSouth studies	
Study 1	Decision-graph programming for service control point services (using the network model)
Study 2	SLL and decision-graph programming of service circuit node services (using the network model)
Study 3	Modifying existing SLL programs, scheduling service invocations, and creating new decision-graph nodes, measurements, and recent change interface (all using the network model)

network, or A-I-Net adjunct-initiated *event*, e.g., "collected digits," "call answered," or "timeout" events. The state-machine structure allows the same type of event to be handled differently at different points in a service flow. Responses to these events typically contain *actions*, e.g., "collect digits," "make a call," or "start a timer," which in turn may lead to other events. The application-oriented language that we constructed for service creation based on the finite-state machine concept is called the *service logic language* (SLL).

In addition to representing call logic, SLL provides service-specific control over OAM&P. SLL supports the formatting of billing records and can define data entry forms and decision-graph structures that can be downloaded to a service management system and populated by service-order clerks or sophisticated end users of a service (e.g., an airline company). Decision graphs consist of a sequence of one or more nodes. Each node either provides a branch to another node or terminates a node. The A-I-Net service creation environment provides a library of nodes that are applicable to many services (e.g., a node that branches the graph traversal according to the time of day). In addition, new service-specific nodes can be defined through SLL service programs. For instance, Panel 2 illustrates a *schedule* node that might be used in a *remind-me* service to schedule a call with a specific announcement to a service subscriber.

We also designed a graphic decision-graph user interface for service management. The service management user interfaces enable customers to tailor service-specific parameters (e.g., a user's personal identification number) and logic (e.g., a call-routing plan that routes calls to different business employees as a function of time of day, day of week, and originating area of the call), respectively.[8]

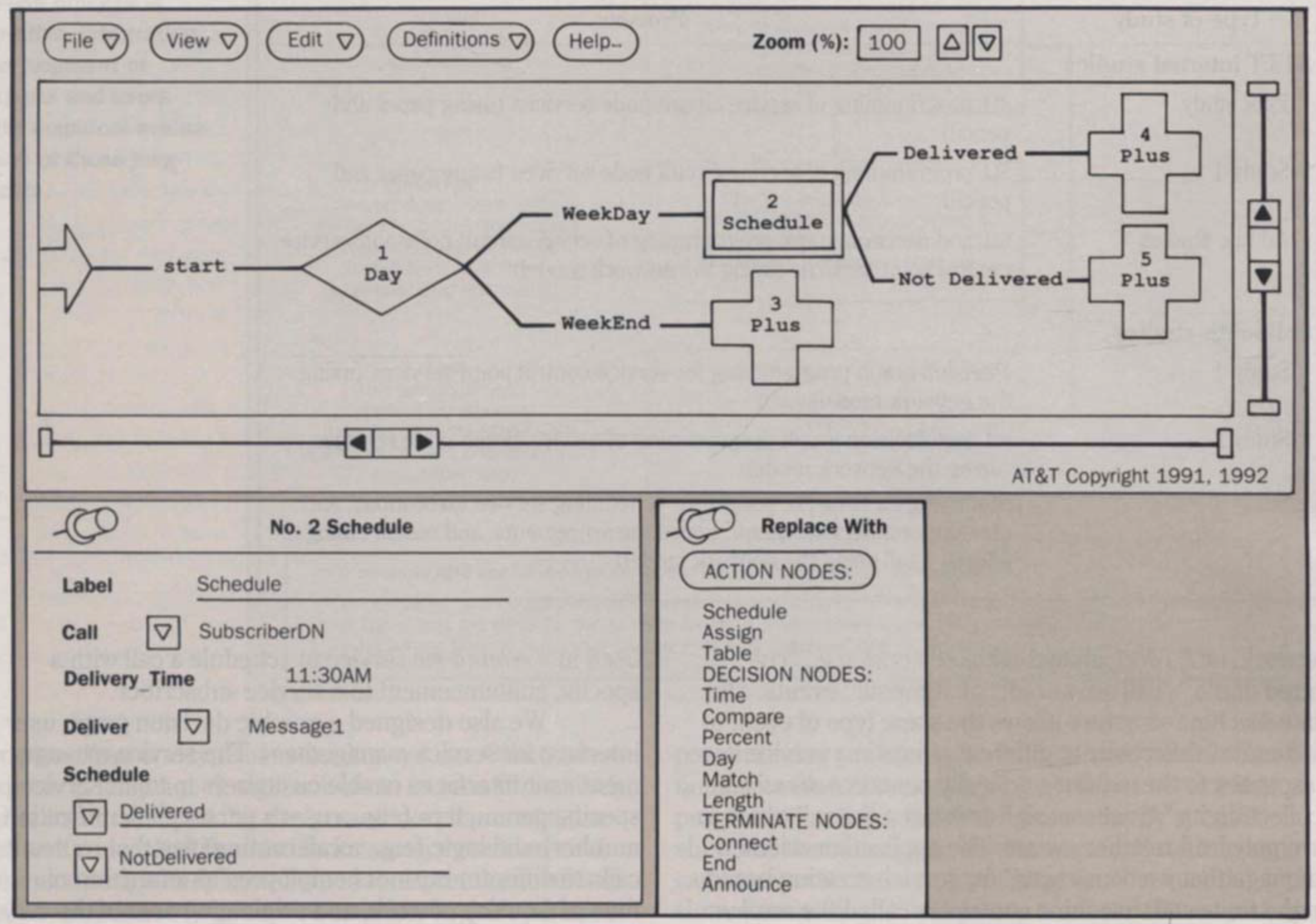
Design-by-Use Studies and Customer Participation

After conducting preliminary discussions with subject-matter experts, we began evaluating our first definition of the SLL. To do this, we:

- Asked service providers to participate in a set of design-by-use studies
- Conducted design-by-use studies with users who had work experience identical to what we expected of SLL users
- Developed an execution environment—a network model—to execute SLL programs and decision graphs and to test services on a physical network. (Figure 3 shows a schematic of the network model.)

Table I outlines design-by-use studies on SLL

Panel 2. Automated Data Flow from SLL to Service Management Interfaces



Label	Schedule
Call	SubscriberDN
Delivery_Time	11:30AM
Deliver	Message1
Schedule	
	Delivered
	NotDelivered

ACTION NODES:
Schedule
Assign
Table
DECISION NODES:
Time
Compare
Percent
Day
Match
Length
TERMINATE NODES:
Connect
End
Announce

```
Schedule         node         action
         {
         Call                  dn,
         Delivery_Time         Time,
         Deliver               Message_enum,
         Delivered             branch,
         NotDelivered          branch}
```

To provide automated data flow, the SLL compiler uses SLL data declarations (e.g., decision-graph node types and recent changeable [rc] variables). These produce data files that can be read by the decision-graph editor and service provisioning forms editor. The decision-graph editor uses the data file to construct graphic nodes, menus, and dialog windows for node parameterization. The service provisioning form displays recent changeables customer data that is service-specific. (This form looks different in the A-I-Net service management system.) To supply a customer with this service, the decision graph user completes the decision graph by replacing the "plus" signs with nodes selected from the "replace with" menu. The service provisioning forms user then completes the service provisioning forms.

and decision-graph programming. Before completing the physical model, we used paper and pencil to conduct a series of SLL programming studies with non-customers (AT&T employees and consultants) who had varying levels of programming experience. After the network model was complete, we conducted ad hoc studies with internal and external customers. In addition, and perhaps of greater importance, we used the network model to conduct a set of studies with the technical staff of BellSouth–Science and Technology.

BellSouth wanted "hands-on" experience with advanced intelligent network concepts by prototyping services of interest to them in a "real" network environment, i.e., the network model. They also wanted to evaluate and understand how the requirements of the service creation environment might affect their work environment. Finally, they wanted more insight into design and implementation trade-offs between services based on the service control point and the service circuit node.

Tutorials. In the common procedure used for our user-evaluation studies, we gave the participants one- to three-day tutorials in a test version of the service creation environment. Following the tutorial, participants received questionnaires and program modification and creation exercises. In the paper-and-pencil studies, we gave participants specifications for services that they were to program. In the ad hoc studies, participants explored the service creation environment using service concepts that interested them. BellSouth participants wrote their own service specifications before the study began. Following some short modification and creation exercises, the BellSouth participants programmed their services for execution on the network model.

Data Collection. The types of data collected were analyses of program designs and code, programming errors, answers to questionnaires, notes taken by participants, and group and individual interviews. The ad hoc studies lasted anywhere from several days to several weeks. The paper-and-pencil studies and the BellSouth studies each required from three to five weeks.

In retrospect, the most successful data collection technique was the group interview held at the end of each week of programming. These discussions included the study participant and members of the design and development team who, together, identified and analyzed problems and proposed solutions. The design and development team then held individual or group discussions

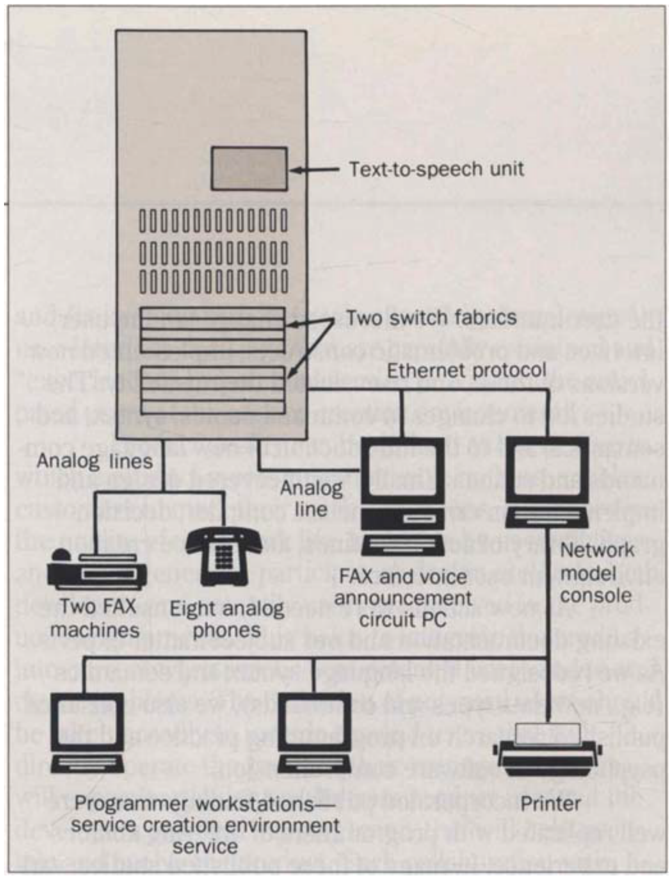


Figure 3. A schematic representation of the network model. Its switch fabrics perform the central office and service circuit node switching function. Together with the network model console, the network model emulates two local switching offices, a service circuit node, a service control point, and a signal transfer point. The network model also emulates integrated services digital network signaling connections between one local office and the service circuit node, and SS7 communication between one local office and the service control point and the signal transfer point. While the network model provides an execution environment for service logic and data generated by the service creation environment, it also allows users to invoke services by dialing from analog phones and enables service logic to ring and answer phones, collect Touch-Tone digits, play voice or text-to-speech announcements, and record, store, and transmit facsimile data.

with the subject-matter experts and the service creation environment/network model development team. Whenever it was possible, proposed solutions were implemented during the study so participants could "experience" them while the problem that inspired the solution was still fresh in their minds.

Design-by-Use Results

In our user-evaluation studies, we gained information about the usability of the language constructs and the user interface. We discussed changes to the user interface and problematic constructs, implemented new versions of these, and re-evaluated their usability. The studies led to changes in command names, syntax, and semantics, and to the introduction of new language commands and actions. Finally, we uncovered design and implementation errors in the SLL compiler, decision graph, library of action routines, and service creation environment user interface.

As new actions were needed, we consulted the existing documentation and our subject-matter experts. As we redesigned the language syntax and semantics (e.g., new data types and commands), we also consulted published research on programming practice and the psychology of software comprehension.

We incorporated published findings that were well replicated with programmers of differing abilities and experience. In many of these published studies, various programming construction, comprehension, and debugging tasks were given to novice or professional programmers. The programmers were asked to use one language or another, where the languages (often created expressly for experimental purposes) differed by only a few dimensions. Published research has shown, for example, that program reliability, comprehension, and maintainability can be increased by requiring logically redundant scope delimiters on conditionals[12,13] (e.g., "if ... endif") and by using assignment statements instead of assignment operators.[14]

These research findings influenced the construction of SLL syntax. No obvious contradictions were uncovered between the research findings used in constructing SLL and the observations gained through the design-by-use process.

In our user-evaluation studies, we found that users of SLL often needed, or wanted, to create nontraditional call states as they constructed service circuit node services. For example, typical telephony call states are ***idle, offhook, digit collection and analysis, routing, ringing, talking***, and ***disconnect***. These call states reasonably reflect the major states of a standard, plain telephone call without advanced telephone features. However, in "Who's Calling" service, in which the name of a calling party is announced to the "Who's Calling" service subscriber before the calling party and subscriber are connected (at the discretion of the subscriber), additional call states (e.g., "caller announced") are appropriate. This important result argues against the industry-wide tendency to standardize and restrict advanced intelligent network call states to a fixed call model. Based on our experience, we believe that new call states will be needed as interface and service circuit capabilities evolve. Therefore, SLL does not restrict the states that can be used in a service program.

We also learned the usefulness of supporting what we came to call flexible logic and its form of expression, a decision graph. Because decision graphs can be created after an SLL program has been compiled and placed in service, they can allow service administrators and even service subscribers to customize their service to a significant degree.

Our work with decision graphs also helped us determine that SLL users needed the ability to define new application nodes. Using SLL, service programmers can create new decision graph nodes, on a per-service basis. They can decide what actions are to be carried out when a node is traversed and what its user-interface characteristics will be when displayed in the decision graph editor.

As a result of the design-by-use studies, we began to appreciate some of the characteristics that made a graphical decision graph editor more usable.

In the decision-graph studies, programmers sequenced decision-graph nodes to specify some aspect of service logic (e.g., a call-routing plan that varies according to time of day and day of week). Except for the nodes that terminate the graph, most decision-graph nodes partition the graph into two or more branches. For example, a time-of-day node might be defined by the decision-graph user as having "morning," "afternoon," "busy hour," and "nighttime" branches. In the early decision-graph studies, the graphic editor did not show incomplete branches (i.e., branches for which no later node was defined). Instead, programmers had to call up a node-specific menu of possible branches. Verifying the decision graph (similar to compiling a program) was valuable for identifying incomplete branches that had not been detected by the programmers. After discussions with user-interface experts and design-by-use participants, we decided to test the decision-graph "plus" notation shown at the top of Panel 2. Test results of studies using this notation were very favorable.

By the end of the studies, we had developed a version of SLL and the decision-graph user interface that was easy to learn and use. New decision graph nodes could be created quickly and reliably. After one week of tutorial and programming exercises, participants could write complex service programs that required from several days to several weeks to code and debug. Because the services created in these studies ran on the network model and did not consider all the OAM&P logic necessary for a deployable service, we must regard this result with caution.

We found that SLL programs could be changed quickly if the person writing the change understood the service flow of the particular telecommunication service. Even people who had no training in SLL could read the program code, and study participants with minor programming experience could learn SLL quickly. Study participants with limited experience in telephony believed that using SLL improved their understanding of telephony.

Combining an easy-to-use service creation language with a network model proved powerful for exploring new service concepts and testing variations in user interfaces to telephone services. The service creation capabilities developed through these studies (i.e., the SLL and the graphic decision-graph interface) provide a core contribution to the A-I-Net product line.

Conclusions

Conducting design-by-use studies has taught us much about the benefits of participatory design practices:

- We confirmed or rejected design assumptions by evaluating usability results.
- We created prototype tools, user documentation, and training materials early in the design/development cycle. However, we caution others that a discrepancy exists between what is easy to prototype and what can be reasonably developed for network-grade deployment. This discrepancy can mislead design evolution and customer expectations. Our subject-matter experts helped us to avoid solutions that appeared wonderful but would be difficult to achieve in a "live" switching office.
- Early in the design/development cycle, we gained valuable insights and sensitivity into user needs and expectations.

Although the process can be time-consuming, we believe that the more effort invested in early participatory design studies, the better the fit between product design and customer needs and expectations. The result can be an effective integration of multiple sources of information and (in keeping with the goals of participatory design[4]) user-interface decisions that are carefully examined and "experienced" by potential service providers (the anticipated users of the service creation environment).

Product quality can be defined as the degree to which a product meets customer expectations, enables customers to meet their work obligations, and improves the quality of their work life. In design-by-use practices and, more generally, participatory design methods, both developers and potential users are involved in the product design process. This helps to integrate the product into current work practices of potential users and to anticipate problems. The definition of potential users should be taken broadly to include the end users that will directly operate the product; their management, who will expect particular results from product use; and the developers and field support teams that will help maintain and evolve the product. Such collaboration with internal and external users of the product (i.e., development and field-support staff and customers, respectively) helps ensure that the product will meet the needs and expectations of its users.

Acknowledgments

The work described in this paper required the help and encouragement of many people. We thank Raymond Bennett, Seymour Bloom, Brian Beuning, Michael Cosky, Phyllis Deppe, Joel Marks, Michael Morgan, Mats Samuelson, Tim Scale, Judy Tschirgi, Ed Weiss, and Lonnie Whitehead of AT&T Bell Laboratories; and Peter Hill, Kathryn Morel, and Daniel Tolosa of BellSouth Technologies.

References

1. J. Whiteside, J. Bennet, and K. Holtzblatt, "Usability Engineering: Our experience and evolution," *Handbook of Human-Computer Interaction*, M. Helander (ed.), Amsterdam: North-Holland, 1988, pp. 791-817.
2. J. D. Gould, "How to design usable systems," *Handbook of Human-Computer Interaction*, M. Helander (ed.), Amsterdam: North-Holland, 1988, pp. 757-789.
3. J. L. Blomberg, "Reflections on Participatory Design: Lessons from the Trillium Experience," *Empowering People: CHI'90 Conference Proceedings*, Seattle, Washington: ACM, 1990, pp. 353-359.
4. G. Bjeknes, P. Ehn, and M. Kyng (eds.), *Computers and Democracy: A Scandinavian Challenge*, Aldershot, UK: Avebury, 1987.
5. M. J. Muller, "No representation without representation: Who participates in participatory design of large software products?" *Reaching through technology: CHI'91 Conference Proceedings*, New Orleans, Louisiana: ACM, pp. 391.

6. R. J. Hass and A. P. Block, "Service Creation: The challenge of the intelligent network," *Proceedings of the National Communications Forum*, Chicago, Illinois, September 1987, Vol. 41, No. 1, pp. 540-542.
7. R. Schwantes, "Network Architecture Evolution: 1990 and Beyond," *Proceedings of the National Communications Forum*, 1987, Vol. 41, No. 1, p. 534.
8. M. J. Morgan, et al., "Service Creation Technologies for the Intelligent Network," *AT&T Technical Journal*, Vol. 70, Numbers 3 and 4, Summer 1991, pp. 58-71.
9. R. Humes, (ed.), "Building the Intelligent Network," *IEEE Communications Magazine*, December 1988, New York: IEEE.
10. Bellcore, *Service Logic Interpreter, Preliminary Description*, SR-TSY-000778, December 1989, Issue 1.
11. O. Faergemand and M. M. Marques, *SDL '89: The Language at Work*, Amsterdam: North-Holland, 1989.
12. T. R. G. Green, "Conditional program statements and their comprehensibility to professional programmers," *International Journal of Man-Machine Studies*, Vol. 9, 1977, pp. 107-118.
13. F. Sykes, R. T. Tillman, and B. Schneiderman, "The effect of scope delimiters on program comprehension," *Software—Practice and Experience*, Vol. 13, 1983, pp. 817-824.
14. J. D. Gannon and J. J. Horning, "Language design for programming reliability," *IEEE Transactions on Software Engineering*, SE-1, No. 2, 1975, pp. 179-191.

(Manuscript received October 29, 1991)

Steven L. Greenspan *is a member of technical staff in the Services and Speech Technology Department at AT&T Bell Laboratories (Indian Hill) in Naperville, Illinois. He works on the human factors and design and evolution of the A-I-Net service creation environment and provides technical support to product marketing and sales. Mr. Greenspan joined AT&T after receiving a joint B.A. in psychology and biology and a Ph.D. in psychology, both from the State University of New York at Buffalo.*

John J. Wisowaty *is a member of technical staff in the Services and Speech Technology Department at AT&T Bell Laboratories (Indian Hill) in Naperville, Illinois. He works on software development for the A-I-Net product line. Mr. Wisowaty joined AT&T in 1984 after earning a B.S. in psychology from Syracuse University, New York, and a Ph.D. in experimental psychology from the University of California, San Diego (LaJolla).*

Raymond E. Bright, Jr., *is a member of technical staff in the Advanced Intelligent Network (A-I-Net) Department at AT&T Bell Laboratories (Indian Hill) at Naperville, Illinois. He is responsible for A-I-Net decision-graph software, works on service creation environment testing, and chairs the A-I-Net service creation environment quality team. Mr. Bright received a B.S. and M.S. in mathematics from Purdue University, West Lafayette, Indiana, and an M.S. in computer science from the Illinois Institute of Technology, Chicago. He joined AT&T in 1979.*