

Rapid Deployment of CPE-Based Telecommunications Services

Wan-Teh Chang, Wei-Yi Li, David G. Messerschmitt, and Ning Chang

Wan-Teh Chang, graduate student, Department of EECS, University of California, Berkeley, California
Wei-Yi Li, graduate student, Department of EECS, University of California, Berkeley, California
David G. Messerschmitt, Professor and Chair, Department of EECS, University of California, Berkeley, California
Ning Chang, Technical Manager, Broadband Services and INA, Pacific Bell, San Ramon, California

Abstract: This paper describes a methodology and platform for the rapid deployment of telecommunications services based on intelligent terminal processors. We propose the *platform approach* to bypass the lengthy standardization process and the community of interest problem. The platforms are intelligent terminal equipment such as workstations with hardware and software resources. Services are defined in terms of platform resources in service description languages and deployed by transferring the service descriptions from repositories to the platforms as a part of the call setup. As a demonstration of this approach, we are prototyping the capability of rapidly deploying services over the Internet based on an object-oriented platform developed at UC Berkeley, Ptolemy, and the prototype uses Ptolemy interpreter files as the service descriptions.

I. Introduction

The rapid deployment of services has been a topic of intense interest in the telecommunications community. Most work in the telecommunications community that addresses this issue, such as the Intelligent Network concept [1][2], has focused on deploying services that are optional enhancements or *features* added to the basic phone service. These efforts assume very basic customer premises equipment (CPE), such as a POTS phone that has primitive signaling capabilities, and have developed enhanced services based on this limited capability. These services typically change or customize certain functions performed by the network, such as routing (e.g. 800 freephone service and call forwarding) and charging (e.g. Alternate Billing Service) without participation from the CPEs.

With the advent of broadband networks and the tight coupling of desktop computing with telecommunications networks, many new telecommunications services based on intelligent CPE with high processing power and continuous media (i.e. audio and video) capabilities will become feasible in the future. A methodology to deploy these new *CPE-based services* quickly and its implication on network software architecture are critical issues that need to be addressed.

In the deployment of new services, telecommunications has suffered from two major obstacles:

- *Standardization.* The traditional view is that because a telecommunications service inherently requires the coordination

of two or more users, all aspects of a service must be standardized so that user terminals can operate in a compatible fashion. The lengthy standardization process adds years to the time required to define and deploy a service. Additionally, because of the compromising nature of the process of standardization and the lack of customer or market experience, the resulting standard is often a poor design.

- *Community of interest (critical mass).* Before two users can participate in a service that requires specialized terminal equipment, they must have anticipated this service by purchasing the required terminal equipment or software. This is a big obstacle to establishing a new service. Early users are discouraged by having very few other users to participate with them. Developers of new services, whether they be telephone companies or third-party vendors, are discouraged by the slowly-developing market that prevents them from recovering their development and deployment investments within a reasonable time frame.

These two obstacles have slowed the pace of innovation in telecommunications and limited the number of services offered. In contrast, we have witnessed in recent years the booming growth of the computer industry. One reason for its fast growth is that, rather than being a specialized gadget with just one application, a computer is a versatile *platform* that can be programmed to perform different tasks such as spreadsheets, word processors, and databases. The platforms are *de facto* standardized after market competition. Stable platforms attract application programmers to develop software that runs on them, and this leads to a proliferation of applications. Another recent success in the computer industry is the file server concept, enabled by high-speed local-area network (LAN) technology. A common set of applications is stored centrally in a file server and can be shared by workstations on a LAN. When invoked, the binary executables are transported from the file server to workstations across the LAN very quickly.

II. An Alternative: the Platform Approach

Inspired by the above two successful examples in the computer industry, we propose the *platform approach* to bypass the two obstacles to the rapid deployment of new services. The platform approach is based on the infrastructure illustrated in Figure 1. The infrastructure has three key elements, which are all standardized:

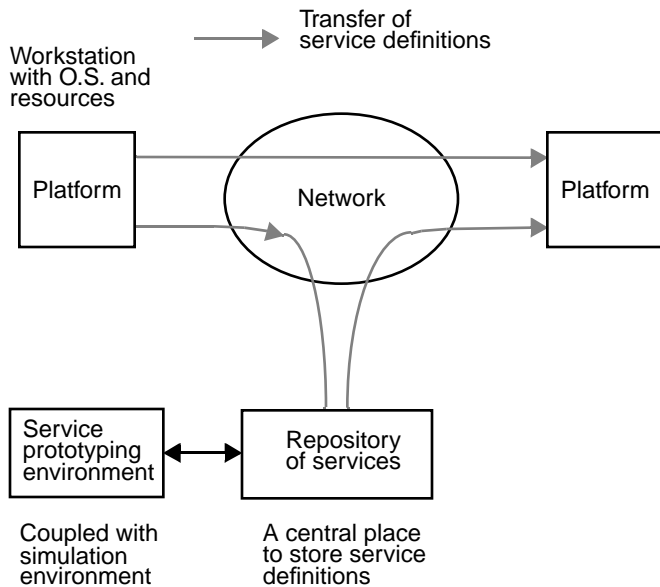


Figure 1. Service definitions are downloaded to the platforms across the network, either from a central repository, or from a user. (Shown in this case are two platforms capable of participating in a variety of telecommunication services. Three or more such platforms could also be used in a multipoint service.)

- A *platform* on the customer premises that contains a collection of resources or primitives for the realization of services. The platform is a capable customer terminal, such as a desktop computer or workstation connected to a telecommunications network, that has adequate processing power for continuous media and service control. The resources can be hardware (programmable processors, custom hardware) or software (OS, software modules).
- A *language* or languages for describing services in terms of the platform resources.
- A *means of transferring the service description* from central repositories to the customer platforms over the network as a part of the call setup. (Alternatively, one of the platforms can serve as its own repository, and the service definition could be transported to the other platform.)

In addition to these elements, it is important that there be rapid prototyping tools and environments coordinated with the platforms and languages to facilitate the development of new services.

The platform approach does not avoid the problems of standardization and community of interest entirely. Rather, the goal is to allow the rapid deployment of new or enhanced services that are based on existing platforms without an intervening standardization cycle and targeted at a community of customers who have the necessary platform resources. Note that standardization is necessary due to the very nature of telecommunications services. Without standardization, the parties would not be able to communicate with each other. But with the platform approach, it is not necessary to standardize each and every new service to be deployed. Only the underlying *infrastructure* needs to be standardized. Standardization is carried out once and for all, and does not stand in

the critical path of the deployment of new services.

The really exciting prospects for the platform approach come with the spread of broadband networking and ever-increasing processor speed. The speed of the network available for transferring a service description will limit the size and complexity of the service description, because a customer will only tolerate a limited delay in initiating the service. Broadband networking should virtually eliminate any constraints on the size of service descriptions. Distributing a large service description with latency acceptable to the user is perhaps one of the most meritorious uses of broadband networks. On the other hand, the seemingly constant increase in processor speed will expand the set of functionality that can be implemented on a programmable basis, and hence reduce the requirement on custom hardware resources. Today voice, audio, and facsimile services can be implemented by programmable processors and software descriptions. In the future, we can anticipate that this will extend to video services as well.

III. Issues

The following issues have implications on the service description languages and the means of distributing service definitions.

3.1 Processor Independence

The service description language should be a high-level language that is interpreted or compiled by the platforms. Otherwise, the service descriptions would not be portable across platforms.

3.2 Performance and efficiency

The platforms should be able to interpret or compile the service descriptions and configure themselves for the services very quickly to minimize the call setup time. Also, the interfaces between software modules should be carefully designed so that the systems dynamically configured can run efficiently.

3.3 Security

Executing an unknown service definition from an unknown source certainly poses a significant security threat. Security is a major issue in rapid service deployment to CPEs. The following two improvements can increase the level of security:

- Authentication of the source of service definition (repository or the other party) at call setup. Common techniques are allowing only certain network addresses/directory numbers, or requesting passwords.
- Limited-applicability description language. We obviously want to prevent things like computer viruses from intruding into customer terminals along with the transfer of service definitions. If the platforms for service definition are general-purpose computers, the users will likely be very nervous about loading executables dynamically into their machines. Hence, it is very likely that the service definitions will not be transferred in the form of binary executables. Security can be controlled much more tightly if the service definitions are in terms of specialized languages designed for that purpose,

which are compiled or interpreted within the service platforms. An additional advantage of this is the compatibility with different processors. The functionality of the service description language should be limited. The language should be free of dangerous commands (e.g. removing files).

IV. Prototype

As a way to demonstrate the feasibility of the platform approach, and to gain more insight into the issues involved, we have been developing a prototype for the rapid deployment of telecommunications services. The goal is to build an environment in which a new service can be prototyped, its operation can be checked out by simulation, and subsequently it can be immediately deployed. We are simultaneously developing several new telecommunications services, which are interesting multimedia applications in themselves, to serve as drivers for our research in service creation and deployment.

4.1 Platform for Our Prototype: Ptolemy

If the platform contains more software resources, then the service definitions will be shorter, and can be transferred with small delay on a slower network. In order to develop a near-term prototype of the rapid deployment concept that is consistent with low-speed as well as high-speed networks, we are using Ptolemy [3] as a platform. Ptolemy was developed at UC Berkeley. It is a simulation and rapid prototyping environment for communications and DSP applications. Ptolemy has a highly-developed base of computational models and functional modules, and systems can be built by specifying the hierarchical interconnection and parameterization of functional modules. The special feature of Ptolemy is its ability to mix heterogeneous computational models (called *domains* in Ptolemy) and design styles.

4.2 Rapid Service Deployment Infrastructure

The prototype of the rapid deployment platform in Ptolemy is shown in Figure 1. In our prototype, each platform consists of a workstation running the Unix operating system and Ptolemy. At present, Ptolemy runs on Sun, Dec, and HP workstations. The resources are the software modules in Ptolemy and hardware accelerators, such as programmable DSPs, supported by Ptolemy. The service description language is the Ptolemy interpreter language called *ptcl*. *Ptcl* is based on the interpreted language Tcl [4] and its associated X window toolkit Tk [5], both of which were developed by John Ousterhout. We made an enhancement to Ptolemy to introduce networking capabilities (for the distribution of service descriptions, for example). The networking code is based on a Tcl extension package called Tcl-DP [6]. Note that the approach we are following here, using the Ptolemy interpreter language *ptcl* as a service definition language, is consistent with the goals of security and processor-independence. For the longer term, we remain cognizant of the security issues.

The network connections in our prototype services use the TCP/IP protocol suite. The choice of TCP/IP is merely for convenience because it is readily available to us on the campus network and the

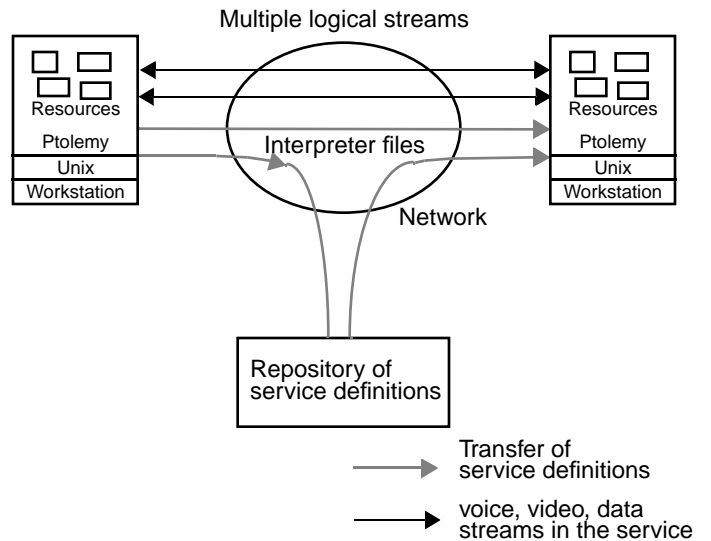


Figure 2. Prototype for the demo of rapid service deployment based on Ptolemy

Internet. The same techniques should apply to network connections using other protocols.

Services are deployed as follows. Shown in Figure 3 is the situation in which the call-initiating party provides the service definition to the called party. We assume that every user terminal has an

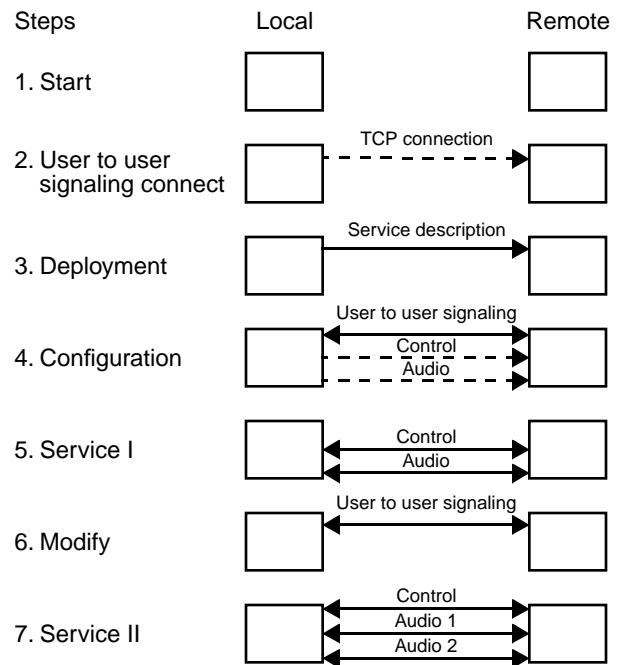


Figure 3. Steps in service deployment

instance of Ptolemy running and listening at a TCP port designated for accepting service descriptions, in the form of *ptcl* files, from other user terminals. At call setup, the initiating party connects to the designated TCP port of the called party, and transfers the service description over this connection. The service description, when executed (interpreted), will configure the user terminals by building Ptolemy systems (called *universes*) with Ptolemy

software modules (called *stars* and *galaxies*), and establishing further network connections to the other party for control (user-to-user signaling) and real-time transfer of data, voice, audio, and video. The user-to-user signaling connection may later be reactivated to modify the running service, or to set up a second service.

4.3 Ptolemy as a Service Creation Environment and Execution Platform

We have also enhanced Ptolemy in the direction of becoming an environment for creating multimedia telecommunications services. We believe that audio and video coding are likely to use international standards. So it is the *choice* or *negotiation* of an audio/video coding standard, *the combination* of different media, and the *service control*, that are most often customized. Fortunately, these parts are not computation-intensive and so their customization can be demonstrated with the current capability of workstations. Also, the control of a service is often very complicated, and formal methods and computer aids will prove useful.

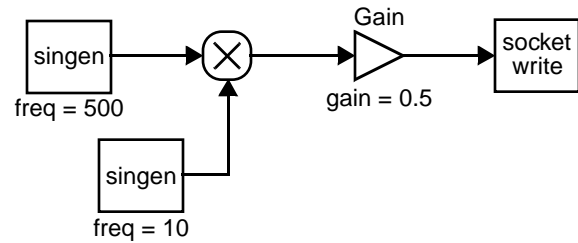
Written in the Ptolemy interpreter language ptcl, each service description may consist of three parts: DSP, I/O (graphical user interface, files, and network), and control. The audio and video processing part may be specified as Ptolemy universes. Dataflow graphs are a natural way to specify DSP algorithms, and Ptolemy has several dataflow domains (synchronous dataflow, boolean dataflow, and dynamic dataflow). The graphical user interface (GUI) is built with Tk. Network connections are set up and managed with Tcl-DP. The control part is currently specified in an event-handler style as a first cut. We make use of the event loop of Tk to handle file and network events and process commands from the GUI. More structured control in the form of hierarchical FSM is also being explored.

4.4 Examples of services

The early prototypes concentrate on services that facilitate collaborative design, and can presently be implemented in software (control and audio). Three “toy” services have been chosen as drivers to shake out the rapid deployment infrastructure in Ptolemy.

The first of our prototype services is the software emulation of voice telephone. It emulates a telephone set with volume control. Each Ptolemy platform (associated with a user) has a microphone for audio input and a speaker for audio output. Two-way audio is transported over TCP/IP networks. Functions of the voice phone are carried out by cooperating Ptolemy universes running on the local and remote platforms, and the Ptolemy universes are bound to the widgets in the GUI. For instance (please refer to Figure 4), to alert the called party of an incoming call, a universe on the local machine generates the ringing tone audio samples and writes them to the endpoint of a TCP/IP connection for network transport. At the other endpoint, a universe on the remote machine reads the audio samples, multiplies them by the gain given by the volume control slider bar in the GUI, and writes them to the speaker. During execution, the called party can hear the phone ringing, and the volume of the ring can be controlled.

The local universe generates ringing tone by modulating a 500 Hz sinusoid by a 10 Hz sinusoid and sends it to a socket.



The remote universe reads the ringing tone from a socket, multiplies it by a gain controlled by a slider bar in the GUI, and plays it over the workstation speaker.

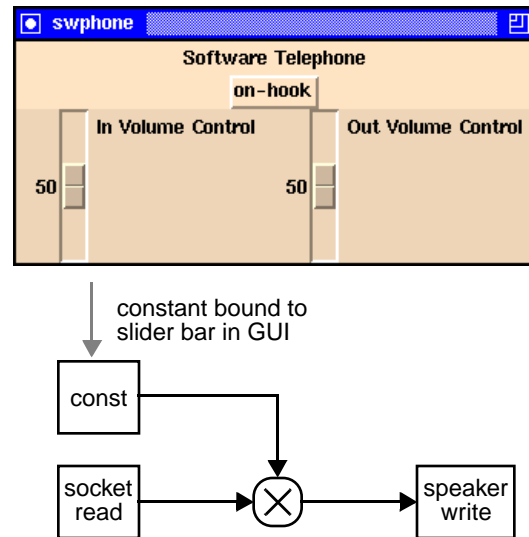


Figure 4. Telephone emulation: alerting

In addition to the telephone emulation, we have also developed two collaborative-design services: a shared drawing board and a collaborative editor. The screen dumps of these services are shown in Figure 5. These services are designed to have a modular architecture, where a central control module is supported by specialized tool modules. All tool modules can be dynamically deployed on demand. Because of the modular architecture, individual tools can be deployed in any combination without affecting the rest of the service.

As the name suggests, the shared drawing board service supports collaborative drawing activities over a point-to-point connection. The central module is built around a Tk-based canvas widget. We relied on Tcl-DP’s distributed objects to provide arbitration and distribution facilities. Supporting tools include modules that let users draw various simple shapes and write short text messages.

The collaborative editor provides more sophisticated shared text-editing capabilities than those provided by the shared drawing board. Like the drawing board service, all editing tools can be in-

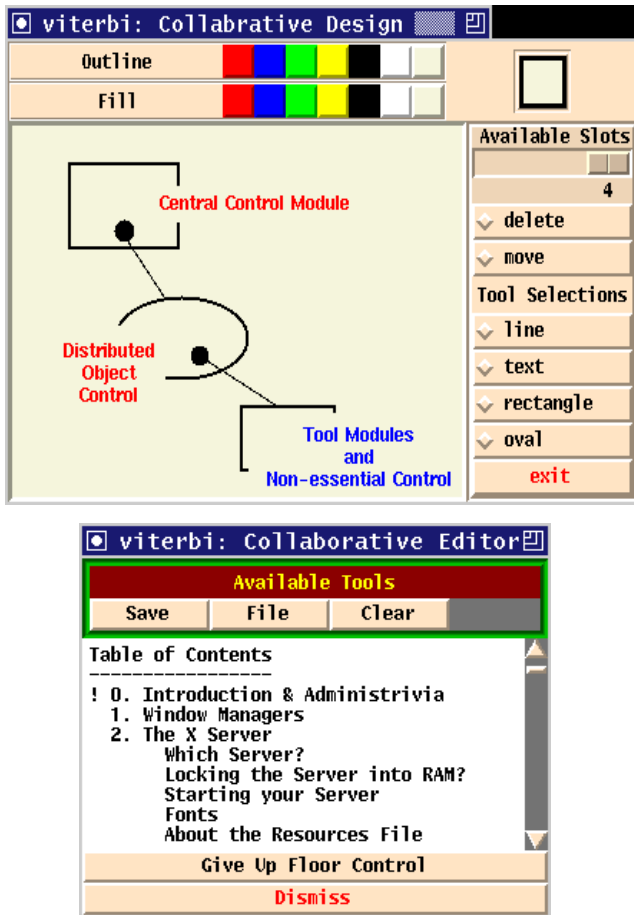


Figure 5. Screen dumps of a shared drawing board and a collaborative editor

dividually deployed, on demand.

We are prototyping some more sophisticated audio-based services in this environment, especially those integrated with existing applications such as voice annotation. In addition, we are also investigating collaborative services over multipoint connections.

V. Related Work

Bellcore's Advanced Intelligent Network (AIN) [1] is representative of the Intelligent Network efforts in the telecommunications community. AIN is a network architecture for the rapid introduction of *network-based services* that provide flexible routing (800 freephone service, call forwarding, etc.), flexible charging (e.g. credit card calling), and flexible user interaction (voice-activated dialing). Since the software to implement these services (called the *service logic programs*) resides in network nodes, deploying these services is essentially "programming the network." The key idea of the AIN architecture is to separate service logic from switching functionality. Service logic programs are centrally stored in special AIN nodes such as Service Control Points (SCPs). The switches, after recognizing that special handling is needed (called *triggers*), query the SCPs for special call processing instructions. New services can be introduced more quickly because fewer network nodes need to be upgraded (just the SCPs, as

opposed to every switch throughout the network before).

More closely related to our work in the deployment of CPE-based services are General Magic's Telescript language [7], and Enabled Mail of Nathaniel Borenstein and Marshall Rose [8]. The idea behind Enabled Mail is that active e-mail messages contain programs that can get executed during delivery, receipt, and display. The choice of their programming language is Tcl (actually a restricted-functionality version called Safe-Tcl) with integrated support for MIME (a format for multimedia messages), display environments, and execution safety.

VI. Conclusions

The platform approach to the rapid deployment of CPE-based services is being prototyped. A new service can be defined in the Ptolemy framework, and checked out there. After it has been verified, other workstations having the Ptolemy platform can immediately participate in this telecommunications service, without the need to standardize the service and without the need for the other users to acquire special software. Rather, the service is deployed by conveying to an instance of Ptolemy on the other workstation a service definition in the form of a Ptolemy interpreter file. The necessary connections are then formed over the network between the two instances of Ptolemy, and the service is up and running.

Because we envision a great deal of functionality within the CPE, our service definition can readily incorporate customized operations on the media signals (e.g. special voice-compression algorithms) and customized commands to the other party (e.g. customized voice mail server). On the other hand, we do not address at all the issue of customized call processing within the network, or the coordination of that call processing with the CPE control mechanisms. These are obvious topics for future research.

References

- [1] R. Berman and J. Brewster, "Perspective on the AIN Architecture," *IEEE Communications Magazine*, vol. 30, no. 2, pp. 27-32, February 1992.
- [2] J. Garrahan, P. Russo, K. Kitami, and R. Kung, "Intelligent Network Overview," *IEEE Communications Magazine*, vol. 31, no. 3, pp. 30-36, March 1993.
- [3] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems", *International Journal of Computer Simulation*, special issue on "Simulation Software Development," January, 1994.
- [4] J. Ousterhout, "Tcl: an Embeddable Command Language," *USENIX Conference Proceedings*, Winter 1990.
- [5] J. Ousterhout, "An X11 Toolkit Based on the Tcl Language," *USENIX Conference Proceedings*, Winter 1991.
- [6] B. Smith, L. Rowe, and S. Yen, "Tcl Distributed Programming," *Proceedings of the 1993 Tcl/Tk Workshop*, Berkeley, California, U.S.A., June 1993.
- [7] J. White, "Telescript Technology: The Foundation for the Electronic Marketplace," *General Magic White Paper*, General Magic, Inc., 1994.

- [8] N. Borenstein, "EMail with a Mind of its Own: the Safe-Tcl Language for Enabled Mail," submitted to *ULPAA 94 Conference*, Barcelona, Spain.