Architecture of the Intuity™ Response Application Programming Interface (IRAPI)

Cary W. FitzGerald
John P. Moosmiller

The Intuity™ Response Application Programming Interface (IRAPI) is a central component in the Intuity CONVERSANT® voice processing system. It provides Intuity CONVERSANT customers with the open interfaces they need to apply an interactive voice response system to their business problems. IRAPI's goals are flexiblility, ease of use, robustness, and interoperability between applications and the extended environment. This paper describes IRAPI, what it enables, how it was built, some of its special features, and its future direction.

Background

The AT&T Intuity CONVERSANT voice processing system,[1] a platform for interactive telephony services, supports many applications both in the AT&T network and on customer premises. The applications developed for an Intuity CONVERSANT system are transaction processing services, that is, services in which people interact with a computer to retrieve information from a database or to request other services. A highly successful AT&T network application of this system is the world's largest deployment of speech recognition for automating operator-assisted calls.[2] A customer premises example familiar to AT&T employees is the benefits registration service. Using this service, employees can select benefit options and receive instantaneous audio feedback and confirmation. With more than 5,000 units in the field, Intuity CONVERSANT systems lead the market in customer premises installations. Version 5.0 acknowledges its link with features of the AT&T Global Business Communications Systems voice messaging products (for example, Intuity AUDIX®) through the name Intuity CONVERSANT system.

Typically, an Intuity CONVERSANT system is connected to both a telephone switching system and a computer host. In this configuration, it acts as a protocol converter, translating end-user voice commands or touch-tone signals into requests to a host computer, database server, or network service. Figure 1 presents a high-level view of this architecture.

Internally, the Intuity CONVERSANT system contains telephony interfaces—such as Integrated Services Digital Network (ISDN)—that connect to a private branch exchange (PBX) or a network toll switch. A critical component adapted from the DEFINITY® Telecommunications System is the time-division multiplex (TDM) bus, which bridges the telephony interfaces and speech processing hardware. Voice processing algorithms, including speech recognition, voice encoding, and playback, execute in real time on special-purpose devices that contain digital signal processing (DSP) chips. Data access is supported using Ethernet and token ring local-area networks (LANs), and also using IBM SNA communication protocols. (SNA is a trademark of International Business Machines, Inc.) Application logic governs the interaction between the caller and the transaction processing host.

Wherever it is practical, the Intuity CONVERSANT product has followed standards for general-purpose computing platforms, such as the Industry Standard Architecture (ISA) bus, small computer system interfaces (SCSIs), and UnixWare operating systems. (UnixWare is a registered trademark of Novell in the United States and other countries, licensed exclusively through X/Open Company Limited.) It has focused develop-

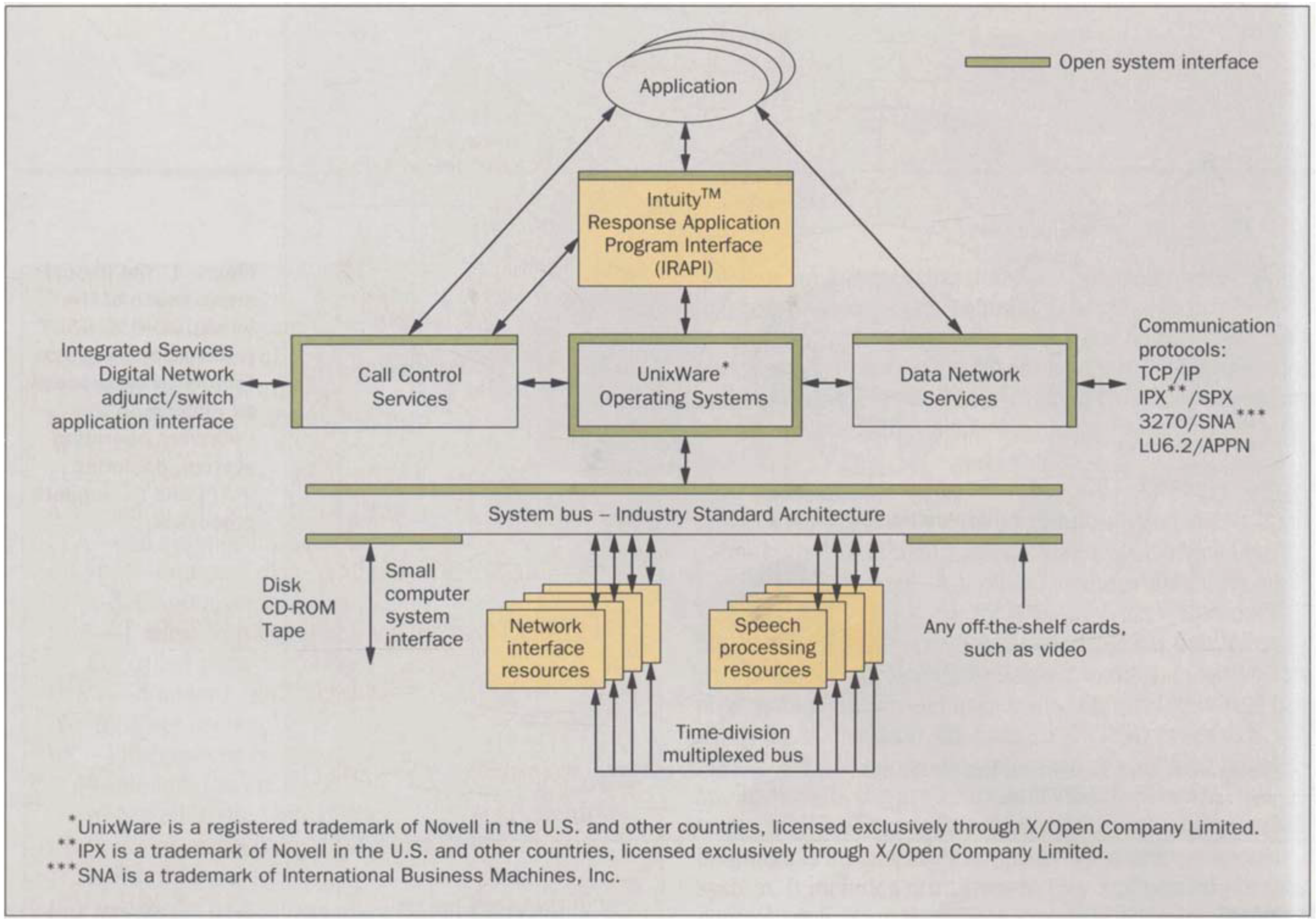

Open system interface
Application
Intuity™ Response Application Program Interface (IRAPI)
Integrated Services Digital Network adjunct/switch application interface
Call Control Services
UnixWare* Operating Systems
Data Network Services
Communication protocols: TCP/IP IPX**/SPX 3270/SNA*** LU6.2/APPN
System bus – Industry Standard Architecture
Disk CD-ROM Tape
Small computer system interface
Network interface resources
Speech processing resources
Any off-the-shelf cards, such as video
Time-division multiplexed bus
*UnixWare is a registered trademark of Novell in the U.S. and other countries, licensed exclusively through X/Open Company Limited.
**IPX is a trademark of Novell in the U.S. and other countries, licensed exclusively through X/Open Company Limited.
***SNA is a trademark of International Business Machines, Inc.

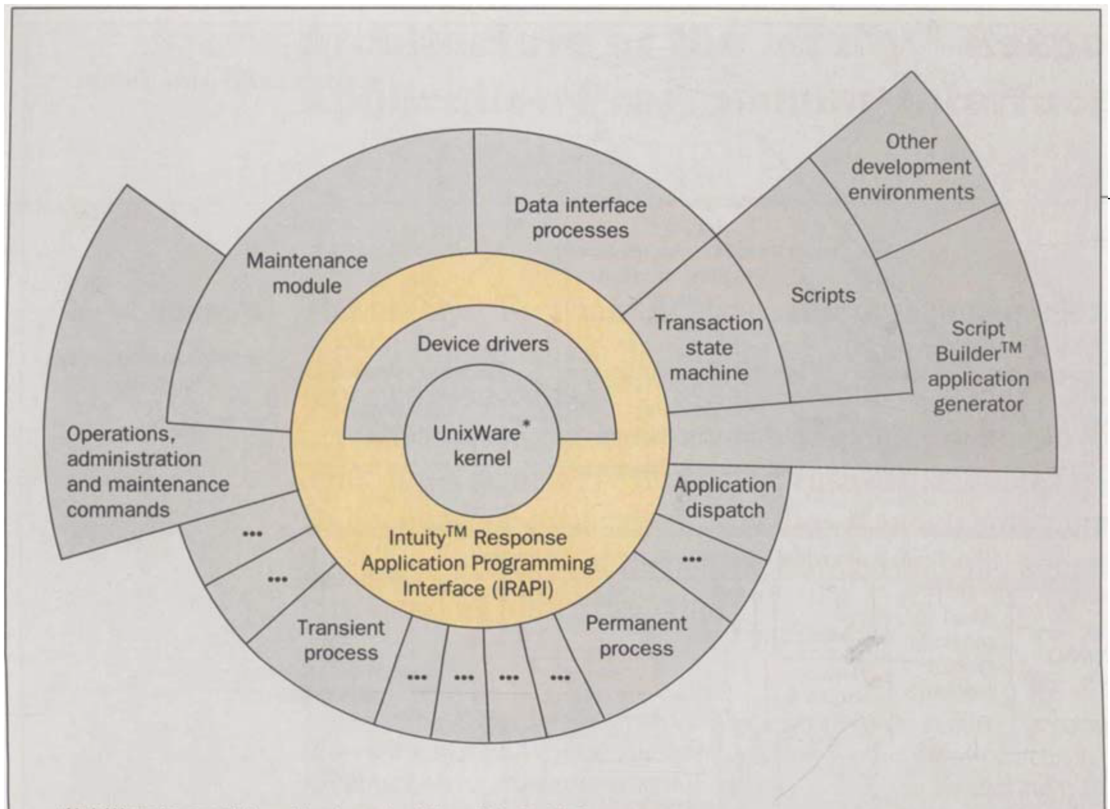
Figure 1. A high-level system architecture showing the open interfaces that exist throughout the Intuity CONVERSANT voice processing system.

ment resources on the core competencies of speech processing and voice system software.

High-Level Intuity CONVERSANT Architecture

The Intuity CONVERSANT voice processing system has always supported application development by development partners in the customer population through tools such as the Script Builder™ application generator. Using it, a developer can quickly and easily build many types of voice-enabled applications. In the past, to maintain system integrity and software quality, development partners were not given direct access to system resources or internal software interfaces. That changed over time, however, as customer needs evolved, particularly those of our most sophisticated development partners. In addition, the needs of AT&T evolved, demanding a new environment—one that is open, modular, extensible, and capable of supporting client/server architectures. A necessary element in such an environment is an application program interface (API) compatible with a general-purpose computing language, in this case, C or C++ programming language. Such an API should address system resources and *events*—notifications when some system condition occurs—at all levels, ranging from devices to application constructs. To protect customers' investments in existing applications, the API co-exists with and supports existing applications and tools for application development.

Technology vendors interested in defining industry standards are also proposing APIs for voice processing. However, Intuity CONVERSANT customers cannot wait for a dominant API to emerge and for robust implementations to be widely available. The Intuity CONVERSANT API is therefore designed to support existing facilities, and to accommodate the importation of new technologies with APIs based on other standards. Version 5.0 is the first major step in product evolution that integrates Intuity CONVERSANT API, known as the Intuity Response API (IRAPI). The reminder of this paper describes IRAPI, what it enables, how it was constructed, some of its special features, and its future direction.


Other development environments
Script Builder™ application generator
Scripts
Data interface processes
Maintenance module
Transaction state machine
Device drivers
UnixWare* kernel
Operations, administration and maintenance commands
Application dispatch
Intuity™ Response Application Programming Interface (IRAPI)
Transient process
Permanent process
•••

*UnixWare is a registered trademark of Novell in the U.S. and other countries, licensed exclusively through X/Open Company Limited.

Figure 2. The logical organization of the Intuity CONVERSANT voice processing system software processes running on the UnixWare operating system, depicting IRAPI and its support processes.

Overview

IRAPI builds substantially on the architecture of previous Intuity CONVERSANT products. Its key requirements are:

- *Backward compatibility.* The overall architecture must not only support C-based applications, but also the existing applications currently being used by Intuity CONVERSANT customers.
- *API ease of use and flexibility.* IRAPI should be flexible enough to support virtually any Intuity CONVERSANT application, yet easy enough to enable any C-language developer to build complex applications.
- *Extensibility.* IRAPI is part of an architecture that allows the Intuity CONVERSANT developers to extend the base platform. In most cases, it should allow extensions to be made without requiring any new releases of the base platform.
- *Robustness.* Because it will be used by developers in the customer population, IRAPI should be able to recover automatically from most errors in applications without affecting other services running on the same platform.

Figure 2 shows the logical organization of the Intuity CONVERSANT platform run time. At the lowest level, the UnixWare kernel supports the file system, application process support, and the framework for device drivers.

Internally, IRAPI manages system resources and information about the state of applications and the channels they own. Utility processes manage the real-time interaction between the file system, the speech cards, and the operations, administration, and maintenance (OA&M) subsystems, the details of which are hidden from the applications. IRAPI offers a well-defined interface that both network interface and resource cards can use to communicate and coordinate with the library, which represents the individual functions that are offered by IRAPI. Standard Intuity CONVERSANT facilities used by IRAPI log errors and participate in the run-time management of the platform.

Requests to invoke any voice system function are channeled through the IRAPI module associated with that function. Because many processes may be interacting with a specific card at any time, IRAPI must associate processes with channels and route messages associated with channels to the application processes that own them. If a channel has no owner, IRAPI will hold its messages in a deferred queue for a fixed period of time. Until some process establishes itself as the channel owner, handling of these messages is deferred. If the channel is not claimed, IRAPI discards the messages and returns the channel to an idle state.

Panel 1. Abbreviations, Acronyms, and Terms

AD—application dispatch
ANI—automatic number identification, that is, the calling party number
API—application program interface
APPN—advanced peer-to-peer networking
ASAI—adjunct/switch application interface
ASR—automatic speech recognition
CCA—call classification analysis
CD-ROM—compact disk–read-only memory
DIP—data interface process
DNIS—Dialed Number Identification Service, that is, the called party number
DTMF—dual-tone multifrequency
IP—Internet protocol
IPC—interprocess communication
IPX—Internet Packet Exchange. (IPX is a trademark of Novell in the United States and other countries, licensed exclusively through X/Open Company Limited.)
IRAPI—Intuity™ Response Application Programming Interface
ISA—Industry Standard Architecture
ISDN—Integrated Services Digital Network
LAN—local-area network
LU6.2—logical unit 6.2
OA&M—operations, administration, and maintenance
PBX—private branch exchange
SCSI—small computer system interface
SNA/3270—Systems Network Architecture. (SNA is a trademark of International Business Machines, Inc.)
SPX—Sequence Packet Exchange
TCP—transmission control protocol
TDM—time-division multiplex
TSM—transaction state machine
TTS—text-to-speech

IRAPI uses a flexible scheme for describing the complex resources available on speech processing resource cards. The packages that implement each function use this scheme to describe the resource to the platform, which frees IRAPI from having to "know" anything about any particular resource. The resource description mechanism ensures that IRAPI will not assign more work to a resource than it can handle. Using a standardized, modular interface, IRAPI also interfaces directly with telephony cards to access telephony and touch-tone services, making it simple to build the logic to connect an arbitrary telephony type into IRAPI.

User Applications

All previously existing Intuity CONVERSANT applications were built using a combination of a special-purpose script language and optionally associated data interface processes (DIPs) running on the UnixWare operating system. (A DIP is a C-language program that follows a particular set of rules that enable it to interact with script programs.) The script language is interpreted in real time using the transaction state machine (TSM) process.

The key to backward compatibility for pre-IRAPI applications is the preservation of the TSM script language and the library interface used by DIPs. The script language is an assembly language for a voice response system. It includes primitives to play and record phrases, control application flow, perform arithmetic and logical operations, maintain an input queue, control advanced speech technology services, and manipulate telephony interfaces. Although the TSM process had been tightly coupled to the Intuity CONVERSANT system software, it was entirely rewritten to accommodate IRAPI. As such, it became an IRAPI application. The TSM does not use any features of the platform that any IRAPI application could not use. The library interface that supports DIPs was also preserved, but its internal function was changed to integrate with IRAPI.

This approach allows Intuity CONVERSANT developers and customers to choose the kind of application control that best fits the problem at hand. The scripts may be generated by the Script Builder application development tool, by building and maintaining applications in the script language using standard UnixWare tools, or by other application development tools. Various customers of the Intuity CONVERSANT platform have used all these approaches successfully; the added ability to write C programs expands the options for application development. IRAPI may also be applicable to user processes, custom applications, and conventional Intuity CONVERSANT DIPs.

Application Dispatch. Application dispatch (AD) is another permanent, multi-channel process. It listens for new calls that arrive on the network, finds an application to handle the call, and starts the associated application. To associate applications with calls, AD uses call information such as Dialed Number Identification Service (DNIS), automatic number identification (ANI), and port number, which is stored in the AD table. Through IRAPI, the OA&M system allows users to add, query, and remove entries from the tables that drive the AD's behavior. Because AD is a standard IRAPI application, customers can substitute an alternative AD that implements another dispatch logic. For example, an alternative AD might not use local tables, but might make the routing decision by consulting a database server over a network.

Debug, Monitor, and Trace. Version 5 of the Intuity CONVERSANT voice processing system contains a number of tools that customers, developers, and customer support engineers can use to monitor and debug the platform and applications. A general trace facility allows users to track the progress of applications and system software. Users can obtain detailed information about the current and past uses of resources. The monitor features can help to debug applications by passively listening to transactions. IRAPI maintains usage information, including current and maximum loads on resource cards. Developers can also use the debug features to simulate unusual and error conditions internal to the platform.

IRAPI Capabilities

IRAPI's high-level, C-language library interface offers an open development interface for voice-telephony applications. Its capabilities include:

- Voice recording,
- Voice storage,
- Voice playback,
- Telephone touch-tone sending and receiving,
- Telephony call progress,
- Speech recognition,
- Text-to-speech (TTS) processing,
- Resource management, and
- TDM bus management (bridging and monitoring channels).

Parameters. IRAPI makes it easy to write applications, while also providing a rich set of options. In any API, these two goals can be in conflict. The design rule that guided the IRAPI specification was to avoid burdening commonly used functions with many parameters to control seldom-needed minor variations on the function's behavior. Instead, minor changes in the behavior of each function are controlled by channel-specific or system-wide parameters, which can be queried and set before the function is invoked. All parameters have defaults suitable for most applications.

Application Structure and Control. IRAPI supports voice applications that can serve many telephone channels simultaneously. Multi-channel applications typically manage each channel of the application independently. Usually, this involves maintaining state information for each channel. Single-channel applications will not necessarily extend to multi-channel applications without difficulty, but multi-channel applications are almost as easy to write as single-channel applications.

Most IRAPI functions that request voice and telephony services are asynchronous, or non-blocking, functions that return control to the application immediately after initiating a service request, rather than blocking control until the service is completed. While the requested service is being carried out, the application can perform other duties, such as servicing a separate telephone channel, accessing a host, or querying a database. When an application is waiting for responses from previous requests, it invokes the IRAPI library, which internally services requests from hardware devices in real time.

When a "condition" occurs, IRAPI notifies the application of an *event*, triggered by any type of condition, and passes control back to the application. In the standard case, applications wait for the requested action to occur in IRAPI before proceeding. Most important, events are generated when asynchronously requested services are completed. Examples of service completions include the completion of voice playback or recording, sending of touch-tone digits, and the timeout of the IRAPI clock.

IRAPI lets an application assign a "tag" to associate an event with a specific service request, such as an IRAPI library call. When voice or telephony services are requested, applications pass IRAPI this tag, which is also included with the event information sent back to the application.

An *interrupt* is the termination of voice/telephony functions when some condition occurs. IRAPI supports a flexible scheme to allow applications to control which events are reported, as well as which conditions will cause interrupts. The application controls most IRAPI interrupts, but some conditions, such as reaching the end of file during the playing of a phrase, terminate voice functions automatically. Internally, interrupt processing for certain events (notably for touch-tone arrivals) is handled as a special case to minimize response time to the event.

Resource Allocation. IRAPI enables applications to access abstract capabilities without having them directly manage the hardware resources that provide those capabilities. IRAPI capabilities include voice record and playback, telephone-call-progress processing, speech recognition, and text-to-speech conversion. Other advanced speech and multimedia services can be added in the future.

IRAPI tries to make resources available whenever the application calls functions that implicitly require resources (for example, "play" or "record"). When a channel relinquishes the resource, IRAPI makes it available to other applications running on the platform. In addition, IRAPI supports reserving, freeing, and querying of resources. If an application must have specific resources, it may pre-allocate them to guarantee that they will be available when needed. If applications must be isolated from each other to avoid resource contention, they can be restricted to a subset of available resources.

IRAPI provides resource allocation failure modes that are consistent for both explicit and implicit requests for dynamic resources. If a required resource is not immediately available, applications can reject the request immediately, wait for the resource forever, or wait for the resource for some fixed period of time. In the last case, if the resource is not available within the time allocated, IRAPI issues an event to notify the application of the failure.

The library uses library states to maintain information about channel activities and to prevent applications from attempting illegal operation sequences. In most cases, applications respond to events as they occur and seldom need to know the library state.

Voice Input and Output. Voice files can be played or recorded to memory buffers and UnixWare operating system files, or into voice file descriptors. Unlike previous Intuity CONVERSANT generics, speech is stored in standard UnixWare file systems instead of in the highly specialized voice file system. To support existing applications and packages that rely on storing phrases in the voice file system, IRAPI maintains the old voice file system semantics using the UnixWare file system. This involves mapping talkfile/phrase numbers to UnixWare files, and vice versa. Voice file descriptors are similar to UnixWare file descriptors.

Speech input and output is controlled by the application, which can stop it explicitly or implicitly using an interrupt. During playback and coding, IRAPI can issue events to notify the application of the progress of the action. Voice file descriptors can be opened, closed, positioned, and converted to UnixWare file descriptors. Applications can query speech files to determine the coding algorithm and convert speech files from one algorithm to another. Internal components of IRAPI are responsible for managing the real-time interface between the file system and resource cards. In most instances, the platform can guarantee that no gaps exist in speech requests. IRAPI includes capabilities to speak numbers and characters with correct inflections. Applications have a similar interface and level of control over text-to-speech (TTS) activities.

Telephony. IRAPI provides basic telephony for a variety of signaling interfaces. Applications can answer incoming calls, place outbound calls—with several options for call classification analysis (CCA)—query, and set per-call information, such as ANI and DNIS, dial dual-tone multifrequency (DTMF) digits, flash, and hang up. IRAPI hides many details of the specific telephony type from the application. If a telephony action is not supported for a given telephony type assigned to a channel, the library will report that the operation is unsupported.

Input Queue and Automatic Speech Recognition. Touch-tones are collected in a unified input queue—also used for automatic speech recognition (ASR) and touch-tone input—which can be manipulated in a variety of ways. IRAPI supports a flexible built-in mechanism for editing input digits, delimiting sequences of digits, timing user responses for the first and for subsequent touch-tone digits, and alerting the application when certain input criteria are reached.

Applications have complete control over ASR. Recognized strings are returned over the touch-tone input queue and, therefore, have access to all the input queue features. Applications can also use echo cancellation to improve recognizer accuracy when speech recognition is required during voice play.

TDM Time-Slot Management. IRAPI provides functions for managing not only the TDM bus, but also network interface connections to the bus. Applications run-

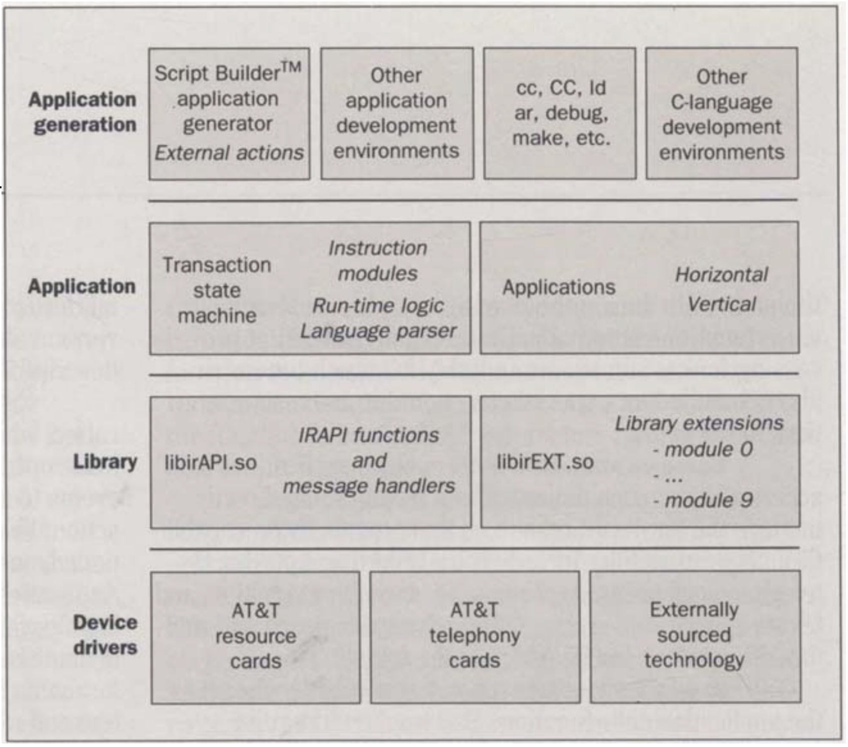

Application generation
Script Builder^{TM} application generator External actions
Other application development environments
cc, CC, ld ar, debug, make, etc.
Other C-language development environments
Application
Transaction state machine
Instruction modules
Run-time logic
Language parser
Applications
Horizontal
Vertical
Library
libirAPI.so
IRAPI functions and message handlers
libirEXT.so
Library extensions
- module 0
- ...
- module 9
Device drivers
AT&T resource cards
AT&T telephony cards
Externally sourced technology

Figure 3. The Intuity CONVERSANT voice processing system is extensible at all layers of the system. Device drivers, library functions, applications, and application generators can all be extended to accommodate new technology resources in productive ways.

ning on several channels can bridge their TDM time slots in a variety of ways. An application can monitor an arbitrary channel, which allows it to listen to all input and output on the monitored channel. A flexible feature allows the application to start and stop background recording, as well as control its volume. Applications can allocate time slots and start activities on them.

Channel Ownership. Applications can negotiate to acquire either specific channels, or one channel in a group. As with resources, applications can choose not to wait, to wait for a fixed period of time, or to wait indefinitely for a channel. IRAPI manages channel ownership internally.

The concept of a default owner is also supported by IRAPI. The default owner is an application that is notified when a channel is freed and there are no other pending requests that this channel will satisfy. Typically, the default owner is the process that is responsible for "listening" for new calls and dispatching applications in response to them (see the earlier description of the Intuity CONVERSANT AD process). Any process can become the default owner for a channel.

IRAPI applications can be *permanent processes* that start and initialize themselves before they are actually needed by any caller, or they can be *transient processes* that are dynamically created only when needed. Any number of applications of each type can either be configured or be actively running on any system.

IRAPI includes a family of functions that allow applications of any type to invoke one another. These functions, which are modeled after the UnixWare exec(2) function, allow one application to replace another from the caller's point of view. This interface is flexible enough to allow IRAPI applications to pass control to TSM-based applications. When processes invoke one another, they can pass information to the invoked process. This facility supports both standard information, such as ANI and DNIS, and user-definable information.

Software Failure Modes and Recovery

Applications using IRAPI may contain bugs. Among other things, applications may request the impossible. They may request services from a resource card that is not on the TDM bus, they may violate IRAPI rules for state changes, or they may pass bad arguments to functions. Because IRAPI cannot depend on well-behaved applications, the library checks the validity of all incoming requests. Invalid requests are rejected immediately, without requiring the library to take any other action.

The application must check for IRAPI events frequently, because the library depends on that mechanism to perform its own time-critical functions. However, some applications may not check for events frequently enough. Under some conditions, application requests will fail if the library is unable to perform its internal functions within the time allocated. Timers monitor time-critical interactions between the library and the voice platform; if the library does not get control in time to perform its function, the system will reject the request and move on. In those cases, IRAPI logs an error and notifies the application.

Some IRAPI applications may contain bugs that cause the application to dump core. The library reacts to applications that abruptly terminate by returning all resources to the free pool, terminating outstanding calls, and cleaning up appropriate data structures to keep other applications that use the channel from being affected.

Dynamic resources are accessible by any application. A system with processes that inappropriately request and then fail to release resources, or an under-engineered system—one with too few resources available to support the mix of applications—will encounter significant contention for resources. Several strategies exist for dealing with this problem: Applications can restrict their universe of resources to a subset of the resources available on the given platform. The library will automatically reject requests for resources that fall outside a previously specified set.

Applications are encouraged to use the implicit resource acquisition/freeing mechanism built into calls for specific requests. For example, a play request will automatically acquire the appropriate resources to play speech and free them when they are done. Resources can be preallocated if they are critical to the application at a subsequent time in the transaction. For example, because some applications find it absolutely critical to have appropriate speech play and voice recognition resources available before they even answer a call, they reserve the appropriate resources beforehand. The drawback to this approach is that applications will hold resources even when they are not using them, making them unavailable to other applications, which may fail as a result. If the application had not reserved the resource, both applications might have been able to share them without difficulty. If resource contention is a problem, applications can use the features described earlier to wait for the resources. Alternatively, more resources can be added to the system.

Some parts of IRAPI use message queues to communicate. If a process fails to read its message queues, it can consume all of the system's interprocess communication (IPC) message queue resources. When this occurs, the system will log errors noting the problem, and the library will protect itself by failing the application's request. In most cases, communication with other processes is protected by timers; if an auxiliary process is unable to communicate with the library, the library will notice that the time has expired, clean up from the request, and continue.

Extensibility

Business pressures have made it imperative to add new capabilities to the Intuity CONVERSANT voice processing system with minimum impact on services, sales, and customers. IRAPI is specifically designed to address this need. The most frequent areas of change are:
- New speech technology and multimedia features,
- New telephony and resource cards,
- New language support, and
- The acceptance of externally sourced technology.

A key requirement is to support the ability to add features to the Intuity CONVERSANT platform, and to support these features at all levels, as shown in Figure 3.

Because the Intuity CONVERSANT voice processing system is based on the standard UnixWare operating system, adding new devices and new device drivers is a feature supported by the UnixWare system itself. New telephony interfaces and languages are supported by specific "hooks," places where software can be added to the library in the future. Device drivers and support software for off-the-shelf hardware components in an Intuity CONVERSANT voice processing system typically come from the hardware providers. New card types and new speech technology features are supported by adding new functions to the library and by adding messages to the message handler routine.

The library component of IRAPI is packaged as a UnixWare SVR4 shared object file. This allows the library to be upgraded without requiring applications to be recompiled. The library includes hooks for integrating new features. The dynamic linking features of shared objects enable Intuity CONVERSANT developers to add new features to the extension library and link them at run time.

Given that both the TSM and AD processes are applications, they can co-reside with, or be replaced completely by, other IRAPI applications. TSM is an example of a meta-application: it runs other applications. The applications it runs have been successful in supporting a particular problem domain, which is a group of problems characterized by a common set of attributes. Developers of Intuity CONVERSANT-based products are free to develop alternative ***horizontal applications*** to support solutions for different problem domains. In contrast, ***vertical applications*** directly solve particular problems. These can coexist and interwork with other horizontal and vertical applications.

At the TSM level, the script language and the TSM interpreter are also designed to be extensible by providing a basic engine for TSM and separate modules to handle parsing the script language and executing the instruction. The model used was the UnixWare idbuild commands. In this analogy, the TSM engine is the kernel and the instruction modules are the device drivers. An analogue to idbuild pulls the whole package together. The development tool of the Script Builder application generator also allows new actions to be added. Extensibility at every level allows the Intuity CONVERSANT platform to absorb new features without forcing fundamental change.

Future Directions

Clearly, IRAPI will be the basis for future development of Intuity CONVERSANT system telephony interfaces, speech technology, multimedia resources, and applications that use them. However, beyond that, there are several areas of active interest in expanding the API and the architecture it supports.

The IRAPI architecture includes a framework for building distributed, client-server applications. The functions of the system have been carefully partitioned to enable users to locate several well-defined functions of the system on remote machines. Application designers will have many options for implementing *extended systems*—voice systems within a larger computing environment. This larger environment may include other Intuity CONVERSANT systems, host systems, special-purpose servers, and desktop systems whose designs will allow users to integrate the Intuity CONVERSANT system more closely and naturally than they have in the past.

The TSM process is just one example of a horizontal application. TSM has been extremely successful in supporting a tremendously wide class of applications that are expressed in terms of the script language. However, substantial room remains for horizontal applications that support other classes of applications in other problem domains. Special-purpose applications that support other paradigms of voice interaction are sure to follow TSM. In the future, the tools and techniques that would allow development partners to add resource cards, telephony cards, and base functionality to IRAPI may become a feature.

One future goal is to develop an API that will support maintenance applications as complex and flexible as those that IRAPI provides for run-time applications. This will be the next step in opening the Intuity CONVERSANT platform to customers who have specific requirements for OA&M systems. It will provide for a variety of "personalities" on the maintenance system, just as alternative TSM implementations provide for different personalities on the run-time system.

Conclusion

IRAPI is a key component in the ongoing movement to provide Intuity CONVERSANT customers with the open interfaces they need to apply an interactive voice response system to the problems they face today. Its goals are flexiblility, ease of use, robustness, and interoperability between applications and the extended environment. Exciting opportunities exist for expanding the role of voice response into new multimedia applications. To a large extent, AT&T's ability to exploit these markets will be dictated by the speed with which new products can be brought to the market or adapted for a specific use.

Karl Martersteck, Vice President of AT&T Network Architecture, defines architecture as "planning for continuous change." This definition guided the specification, design, and implementation of IRAPI. Accepting change is a fundamental capability of the platform, at both the application and platform levels.

References

1. R. J. Perdue and E. L. Rissanen, "AT&T Voice Processing System Architectures," *AT&T Technical Journal*, Vol. 69, No. 5, September/October 1990, pp. 52-60.
2. W. E. Longenbaker, R. J. Perdue, and S. M. Salchenberger, "Automation of Operator Services: A Successful Application of Speech Recognition Technology," *Proceedings of the Second IEEE Workshop on Interactive Voice Technology for Telecommunications Applications*, September 26-27, 1994, Kyoto Research Park, Kyoto Japan, pp. 161-164.

(Manuscript approved February 1995)

***Cary W. FitzGerald** is a member of technical staff in the Voice Transaction Systems Department at AT&T Global Business Communications Systems in Columbus, Ohio. He works on system architecture for the Intuity CONVERSANT voice processing system. Mr. FitzGerald joined AT&T in 1992, after receiving a B.S. in computer science from Purdue University, West Lafayette, Indiana.*

__John P. Moosmiller__ is a distinguished member of technical staff in the Voice Transaction Systems Department at AT&T Global Business Communications Systems in Columbus, Ohio. He is responsible for planning system architecture design of next-generation multimedia transaction servers for telephony and desktop workstations. Mr. Moosmiller received a B.S. in biochemistry from Purdue University, West Lafayette, Indiana, and an M.S. in computer science from Ohio State University, Columbus. He joined AT&T in 1983.