AIN System Development—A Customer-Centered Approach

James T. Smith
iamjts2@yahoo.com

Abstract: A high-level view of an information model and system architecture for the AIN—the Advanced Intelligent Network—is presented which places the customer as the focus of all service design, development, deployment, and management. In this paper, the service context profile—one of the major components of this architecture—is used to indicate how several technologies from the general field of distributed artificial intelligence (DAI) may be applied with far-reaching consequences.

1. The Proposed Model -- a First Pass

In this section a top-level logical view is presented of the proposed design and information model, prior to focusing upon the service context profile. A natural first decomposition of the information required by the proposed AIN architecture is based upon the differentiation of that information which is customer-specific, that which is system-specific, and that which is the expression of various relationships between information from the other two. The major components are:

- Service Subscription
- Customer Context
- Service Context
- Service Description
- Service Management
- Service Control Loop

The left column represents general categories of customer-specific information and processing. The right column represents categories of AIN system-specific information and processing.

1.1. Top-level Service Management

A pair of coordinated, complementary top-level system managers—a service administration manager and a real-time feature manager—is assigned to the customer, as a result of the subscription process. This assignment is reviewed each time the customer's subscription is modified.

The ***administration manager*** (**AM**) controls the customer's modification of information within the customer context and service context profiles. In particular, the AM is to facilitate the customer's administration of his services and features so that he receives maximum benefit—a subjective call—from the positive interactions of his service features, and yet to prevent a configuration that would result in negative, or conflicting behavior.

The ***feature manager*** (**FM**) controls the real-time execution of the customer's various services and their interactions with each other and with the system, as specified by the customer subscription, customer context, and service context profiles. Based upon the customer and service context profiles, and upon AIN system state information, the FM is responsible to

- Analyze the current contextual state of the customer's services at any given moment,
- Select appropriate service actions, and
- Manage their execution.

The AM and FM are *coordinated* in the sense that the FM must be able to manage the real-time execution of whatever service behavior the AM has specified; while, the AM must be able to detect and prevent the customer's specification of situations which the FM would be unable to manage. The two managers thus work in a coordinated manner to provide the customer with a richer set of service capabilities.

In each instance of service execution, the information in the customer context and service context profiles (described below) is processed to determine what is the most appropriate action to be taken by the system. The two managers are *complementary* in the sense that the FM's real-time processing not only is moderated but also is optimized by appropriate pre-processing by the AM.

An important adjunct functionality of these two managers is *feature interaction negotiation management*. This is the process by which the AIN system, in conjunction with the AM and FM, determines *acceptable alternative* service behavior, whenever the service context of other involved parties, or the capabilities of other AIN systems are not consistent with the *preferred* service behavior of the customer whose services are being managed.

1.2. Service Description Profile

The ***service description profile*** provides a comprehensive top-level *declarative* view of the AIN system. The information in this profile is *comprehensive* in that it contains descriptions of all service features and capabilities offered via the AIN system. The information in this profile is *declarative* in that the information provided here makes no assumptions regarding the procedural *implementations* of those services.

1.3. Service Management Profile

The ***service management profile*** associates the features and capabilities described declaratively in the service description profile with specific implementations within the AIN system, the specific platforms—e.g., a specific IP, intelligent peripheral—where they reside in the network, the specific resources they require—e.g., access to a customer's voice-prints—etc.

The service description and service management profiles together represent the abstraction of the specification of a service's behavior from the implementation of that behavior. The former contains the knowledge needed by the intelligent AIN system to support both the non-real time service management and the real time service execution functions provided by the latter.

1.4. Service Control Loop

The ***service control loop*** provides the real-time framework for the management of the customer's services. At the top-level of each customer's services execution, the FM of the customer manages the execution of the customer's services, being driven by information in the customer's service context profile (which in turn is dependent upon the service description profile, the service subscription profile, and the customer context profile), real-time state information, and the system's service description and management profiles.

Similar to the blackboard paradigm of DAI (distributed artificial intelligence), the FM *evaluates* the customer's service context profile, along with system state information, to *resolve* (determine) the currently *focused context*. From this context, the FM selects the next action to be taken, i.e., which service action is to be executed. From the service description and management profiles, the FM identifies which *implementation* of that service is to be invoked. The FM repeats this *resolve-context, select action, execute* cycle until a *completion* condition is identified and successfully executed.

1.5. Service Subscription Profile

The information stored in the ***service subscription profile*** specifies the combination of AIN services and features, including management features, billing options, etc. which comprise the top-level of controls and constraints for how that customer wishes his services to be managed and delivered. The customer may identify not only *preferred* services and features but also any *alternate* feature behaviors regarding the services which are subscribed. This information is used in support of *feature interaction negotiation* as needed during the system's delivery of his services.

1.6. Customer Context Profile

The ***customer context profile*** contains customer-provided *service-neutral* contextual information that is needed to drive (specify) the execution of the customer's services. Its management is under control of the customer's AM. The intent of this profile is to provide a consistent, customer-centered mechanism for the customer to describe the various contextual elements which he can use to indicate the circumstances under which each subscribed service and feature is to be invoked. It includes personal-contact information such as the identities of individuals and of groups of people—e.g., *"My Doctor"* and *"Sales*

Accounts," _temporal_ information such as periods of time—e.g., *"Business Hours"* and *"Special Operations,"* and _customer-state_ information—e.g., *"Do Not Disturb"* and *"VIP's Only."*

The information stored in the customer context profile is _*service-neutral*_, that is, each item of information in it may be applied in the specification or qualification of the behavior—e.g., feature options—of any combination of the customer's services. While shared by all services, this information is uniquely managed within this one profile. Any modifications made to this profile immediately affect all service context specifications that are based, directly or indirectly, upon the modified information.

*1.7. Service Context Profile*

The ***service context profile*** specifies the contexts under which each subscribed service feature and management feature is applicable to the delivery of the customer's services. Its management is under control of the customer's AM. The customer uses information from the customer context and the service description profiles in the generation and the administration of service contexts.

Information from the customer context profile is used to provide the *"whenever's,"* the *"with whom's,"* the security PIN's, customer states, etc. which are needed to specify a _*service context*_. Information from the service description profile is used to identify the logical system states—e.g., *"Line-busy"* or *"Transfer-in-progress"*—for which a particular service behavior may be specified.

While such logical system state descriptions may correspond—even one-to-one—to system specific information, such as that provided via specific AIN call-model triggers, their meaning and use in these contexts are not dependent upon the system's mechanisms for determining this state information.

Although AIN call-models, etc. may evolve, service implementations may be modified, and such mappings may change; still, the semantics *of specification of service context and behavior by the customer are to remain* consistent *to the customer.*

2. An Example of a Customer's Service Context Profile Is Examined

An example of a customer's *logical view* of his service context profile—in *spreadsheet format*, for purpose of this discussion—is shown in Table 1. Although many details have been simplified or omitted, this example can be used to illustrate not only various aspects of the service context profile, but more importantly to indicate several far-reaching ramifications of this total AIN system architecture.

The first four columns indicate _*service-neutral*_ contextual information. The contents of the first three columns are _*user-defined*_ in the customer context profile; that of the fourth column is _*system-defined*_ in the service description profile. The last three columns indicate the _*service feature behavior*_ desired by the customer for the contexts indicated by the first four columns.

The entries under the System *State*, *Feature*, and *Option* columns are _*system-defined*_ within the system's service description profile which defines all services, all system states, and in particular identifies those states meaningful for each given service and feature. Such terms as *"WorkHrs," "Boss," "Spouse,"* and *"Sect"* are _*customer-defined*_ in the customer context profile. Such terms as *"FCF," "CND," "TimeOut,"* and *"Alert"* are _*system-defined*_ in the service description profile.

The appearance of an asterisk "*" in this example indicates an *"Otherwise,"* or *default* value; while, the question mark "?" indicates a condition in which the system has not determined a value for this item. The last entry is an example of a context that implicitly supports _*feature interaction negotiation*_:

*	?	*	*	FCF	Request	ID-Msg

This entry indicates the service action to be taken in the event that the calling-party is not known. This service context entry indicates an alternative service behavior of the customer in the event that CND is blocked by the party calling the customer! This entry is an example of the customer's service context profile used to specify feature negotiation *fall-back* positions on an individual, case-by-case basis.

Time	Who	Cust.State	Sys.State	Feature	Option	Value
WorkHrs	Spouse	*	TimeOut	FCF	Fwd2	VM-Wife
WorkHrs	Boss	*	TimeOut	FCF	Fwd2	Sect.
WorkHrs	Boss	*	Busy	FCW	Alert	Quiet
WorkHrs	Boss	DoNotDisturb	*	FCW	Alert	Quiet
WorkHrs	Sect	*	Busy	FCW	Alert	Loud
Late	*	OnCall	TimeOut	FCW	Alert	Loud
Late	*	OnCall	Busy	FCW	Alert	Loud
Late	*	DoNotDisturb	*	FCF	Fwd2	VM-Std
WorkHrs	*	*	TimeOut	FCF	Fwd2	VM-Std
*	*	Meeting	*	FCF	Fwd2	Pager
*	Special	Meeting	*	FCF	Fwd2	UptNum
Lunch	*	*	*	FCF	Fwd2	VM-Std
Late	Wife	*	TimeOut	FCF	Ask	PageMe
*	*	*	*	CND	Alert	*
*	?	*	*	FCF	Request	ID-Msg

Table 1   An Example Customer Service Profile

3. Some Far-Reaching Consequences of this Design

This AIN information model and system architecture facilitates the partitioning of information management and feature processing both vertically into islands of independently implementable functionality that are highly reusable and interchangeable, as well as horizontally into layers of increasing abstraction that provide tremendous flexibility in how the customer's services are managed and delivered.

The example of Section 2 is used to indicate several important aspects or consequences of this AIN architecture. which are summarized here, and will be discussed in the sections that follow.

- Customer-Centered Service Management — The customer is provided through the use of labels (symbols) that are meaningful to him with the ability to interact with the AIN at a declarative level that does not require procedural knowledge of how a feature is or could be implemented. The internal workings of the AIN are completely encapsulated from the customer.
- User Interface Synergism — Multiple user interfaces (UI's) are supported—e.g., textual, vocal, DTMF/IVR, and graphical—for the management and execution of the customer's services. They all share a consistent *"look-n-feel."* In particular, changes to the behavior of one UI result in corresponding changes to the behaviors of the others. Advanced multi-lingual pseudo-natural language interfaces, including scripting capabilities, also are supported by this architecture.
- User Information Synergism — All customer-provided information is enterable once (per change) and in one unique place, in a consistent manner that is independent of which, of how many, or of what types of services are, or will be subscribed which might use or be affected by that information.
- Normalization of Information & Processing — Information management and feature processing are partitioned vertically into islands of independently implementable functionality. The various types of *service-generic* information—day-timer, phonebook, customer states, customer contexts, etc.—which the customer may specify are implemented so as to be administered and processed independently of each other in a plug-n-play manner.
- Symbolic Declarative Information — Information management and feature processing are partitioned *horizontally* into layers of increasing abstraction and generalization. The customer is provided with the ability to state his problems—desired service behaviors, and their solutions—in customer-understandable terms—in as abstract and as general a manner as possible.

- Flexible Processing Strategy — The symbolic declarative representation of the customer's service management information is neutral to the particular realization of any specific service feature. This provides the basis for a virtually unlimited flexibility as to what services such information may be applied, and as to how and when that information may be utilized.
- Information Abstraction & Late-Binding Interpretation — The declarative symbolic information of one layer represents an abstraction (model) of the processing (procedures) of lower functional layers. Under different real-time circumstances, the same information abstractions may be mapped to different realizations, and may be applied in different ways to satisfy quite different requirements. Information abstraction leads to information reuse.
- Feature Interaction & Negotiation Management — The customer's profiles are treated by this AIN architecture as representing the customer's *preferred* service feature behavior, rather than as the expression of a set of rigid specifications. The goal of the FM is to select the service contexts which better satisfy the customers *preferences* and also are compatible with real-time system states and capabilities. In particular, the FM's of interacting customers utilize these symbolically expressed *preferences* as the basis for *negotiation* of service behavior that is acceptable to those parties.

Each of these major points are now elaborated in greater detail. These points are cumulative, that is, the functionality's pertaining to the latter points build upon those of the earlier points. Effort is made to preserve this continuity without being redundant. Additional discussion of these points appears in other papers of the author, per the references.

3.1 Customer-Centered Service Management

The customer is provided through the use of labels (symbols) that are meaningful to him with the ability to interact with the AIN at a declarative level that does not require procedural knowledge of how a feature is or could be implemented. The internal workings of the AIN are completely encapsulated from the customer.

In most current AIN systems, information provided by the customer for the management of his telephone services must be entered and administered in a form that is *natural* to the service—i.e., the service designer, rather than *natural* to the customer. For example, speed calling lists, call screening lists, etc. typically are administered as independent lists of phone numbers only. Their association with the *reasons* why they appear in each of these lists—e.g., home, parents, doctor, work—is not supported by current information models. Similarly, intervals and periods of time used to constrain service behaviors are divorced from their *meaning* to the customer.

In contrast, the customer here is able to *associate*—and thereafter to use—customer-meaningful *labels*—e.g., *"Family," "Dr. Jones," "Business Hours"*—with any information that she might provide as depicted in Table 1. Since such labels are not required by internal service execution *logic,* the AIN information model organizes these labels distinct from, yet linked to, the internal system information with which they are associated in such a manner that real-time performance of the service is not compromised.

As a first approximation of its implementation, a *label* (or symbol) could be simply a *tuple*, for example (text string, internal ID). The textual component may be initialized and modified by the customer; while, the internal ID is assigned and managed by the system to reference various entities, such as an associated phone number, etc. The customer can establish the association of a given label with some entity—e.g., a value, procedure, or process. Furthermore, the customer may use these labels to *symbolically* express relationships, independent of the particular associated entities.

The purpose of this *label mechanism* is to enable the customer to use customer-meaningful terms that are natural to him to express any conditions and aspects about the behavior of his telephony services. Not only is the customer provided a more user-friendly way to manage his services; more importantly, he is empowered to interact with the management and the execution of his services at levels of abstraction not previously realized with current systems.

3.2 User Interface Synergism

Multiple user interfaces (UI's) are supported—e.g., textual, vocal, DTMF/IVR, and graphical—for the management and execution of the customer's services. They all share a consistent "look-n-feel." In particular, changes to the behavior of one UI result in corresponding changes to the behaviors of the others. Advanced multi-lingual pseudo-natural language interfaces, including scripting capabilities, also are supported by this architecture.

The label construct represents the focal point of the customer's interaction with the management of his services. To generalize the simple implementation of a label mentioned above, a label consists of two logical components: 1) a *system-side component* (i.e., an internal ID) to reference and access system-side objects, processes, etc., and 2) a *customer-side component*, which may have several different supported UI realizations—e.g., textual, vocal, iconic, pictorial—dependent upon the customer's subscriptions. For example, the text string *"Family,"* the vocal sound-bite *"Family,"* and the *"Family"* icon all map to the same *Family* label.

Furthermore, the various customer interfaces for the management and manipulation of these labels maintain a *coordinated behavior* as the customer configures and uses them. For example, the menu hierarchies associated with the customer's *menus* on his PC-based interface and his DTMF/IVR interfaces provide *equivalent* navigational paths to the same functionality. Quick-key short cuts on the PC interface coincide with #,*-key short cuts on the DTMF/IVR interface. Association of the F5 key of his PC interface with a certain action results in the corresponding DTMF/IVR action being associated with the #5 key, and with the "pound five" phrase. Thus visual, verbal, and tactile navigation of the service management functionality are always *consistent*.

Not only is the customer's service administration customer-centered, so also are all supporting AIN operational functions. For example, billing of the customer's services is based on the same terms that are used by the customer to administer her services. Figure 1 below emphasizes the pivotal role that the label, as the foundation for symbolic abstraction, provides to this AIN architecture.

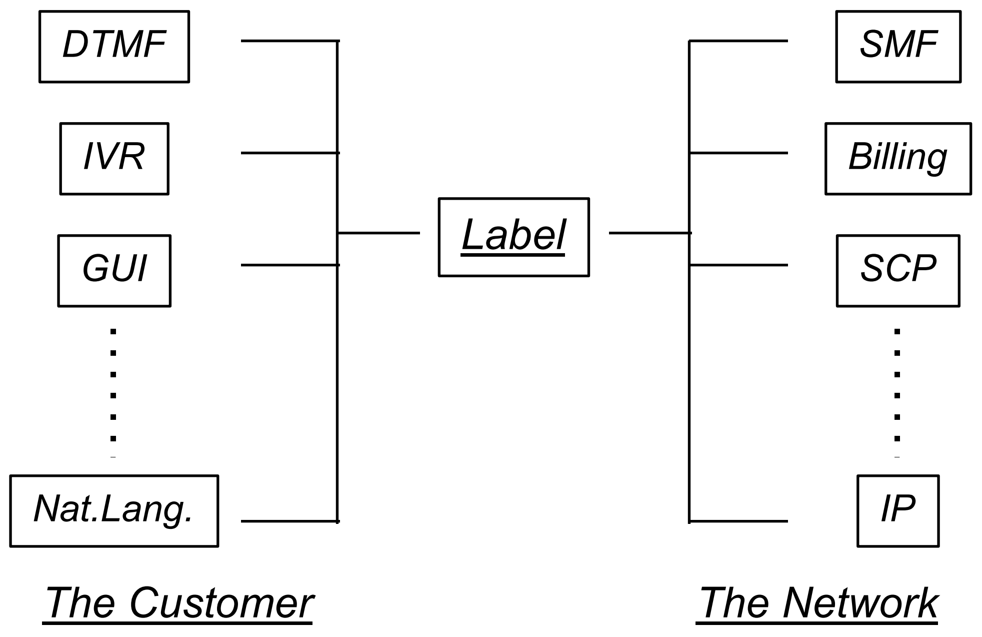

Figure 1 Customer-Centered = Symbol-Centered

While the *spreadsheet paradigm* typified by the example of Table 1 provides a powerful and concise view of the service context profile, it is by no means the only presentation format that the customer interface might take. The customer-defined labels, together with system-defined labels, provide the foundation for the system's support of both: 1) a *multi-lingual pseudo-natural language* capability for the customer's self-administration of his services, as well as 2) a *scripting language* capability for the creation—or packaging—of *customer-defined* service behaviors formed from *system-defined* service feature capabilities.

The Label system provides the foundation for the implementation of the *pseudo-natural language* expression of customer definitions and service contexts. As examples, the customer might express the following statements as she modifies her customer and service context profiles:

Add *Johnny* **to** *Friends*

New Individual *MyFax* **IS** *214-718-6398*

and

If *Work* **And Fax Then Fwd2** *MyFax* **And AlertMe** *MyPager.*

In these examples, the boldfaced labels reference *system-defined* entities, and the italicized labels reference *customer-defined* entities.

Grammar rules for this pseudo-natural language are determined from the customer's subscription. The service management features define the domain of permissible statements which the customer may make. In the first example, the customer adds an *individual* named *"Johnny"* to a customer-defined *group* named *"Friends."* Next, the customer identifies a fax machine phone number to be associated with the *individual "MyFax."* Finally, the customer requests that any faxes received at his service number—such a determination is a *content-based* service—be forwarded to this fax machine, and that he be alerted via his pager—which effectively specifies a composite *customer-defined* service behavior.

Multi-lingual versions of system-defined labels are supportable by the AIN. During service subscription, the customer may select his language of choice from those that are supported. To support additional natural-like—English-like, French-like, etc.—languages is a matter of mapping appropriate words and phrases to the system-defined symbolic labels which provide the interface between the various UI's and the various AIN systems, and possibly of reordering their expected positions within phrases—new grammars.

The following are implementations of the previous examples as they might appear in Spanish.

Sumar *Juan* **A** *Amigos*

Persona Nueva *MiFax* **Es** *214-718-6398*

and

Si *Trabajo* **Y Fax Entonces Transferir** *MiFax* **Y AlertMe** *MiBeeper.*

3.3 User Information Synergism

All customer-provided information is enterable once (per change) and in one unique place, in a consistent manner that is independent of which, of how many, or of what types of services are, or will be subscribed which might use or be affected by that information.

Service-generic information—which could be applicable to many different service contexts—is managed independent of those services and features which might utilize that information. Once such an item has been defined by the customer, it is available for use in constraining or modifying the behavior of any service feature. Such information includes, for example, that normally found in a personal information manager (**PIM**), e.g., a day-timer and a phonebook.

Temporal entities are an example of feature *constraining* information. The execution of a feature need not require the time of day; however, the customer may wish that feature to be considered for execution only during a certain time period. *Modifying* information identifies values, processes, etc. needed for a feature to complete, for example, the phone number to which a call-forwarding feature would redirect a call.

As an example, were the customer to modify the meaning of *"WorkHrs"* used in Table 1, then every entity of the customer's service management that is dependent upon the meaning of—associated with—

"WorkHrs" is correctly affected—e.g., service context rules that refer to *"WorkHrs,"* and any other time periods that are built upon *"WorkHrs."* Similarly, a given customer-defined list of phone numbers—e.g., the *group "PreferredAccounts"*—could be used to affect multiple feature behaviors under different circumstances—e.g., *"RegularHours"* versus *"WeekendHours."* Such information is entered by the customer, managed by the system, and utilized during real-time in a consistent manner.

Dependent upon the sophistication of the customer's subscribed service management, quite complex service-generic abstractions can be developed (expressed) by the customer. As examples, are the individual members of a group always explicitly identified as members of a given group, or may they be *inherited* by virtue of membership in another group of individuals that is inherited by the given group? Are customer states—e.g., *"DoNotDisturb"* and *"OnStandby"*—to be treated as mutually exclusive—i.e., setting one state to *ON* automatically clears all others to *OFF*, or may logical AND's and OR's of states be specified—either explicitly as a defined composite state, or implicitly by the juxtaposition of two or more states within the specification of one service context?

3.4 Normalization of Information & Processing

Information management and feature processing are partitioned vertically into islands of independently implementable functionality. The various types of service-generic *information—day-timer, phonebook, customer states, customer contexts, etc.—which the customer may specify are implemented so as to be administered and processed independently of each other in a plug-n-play manner.*

One implementation of a given service-generic functionality, say the *day-timer*, may be replaced with another implementation, without affecting the customer's existing management of his service context profile—which utilizes the day-timer—or the execution of the service features which utilize such information. In the service context profile example of Table 1, the *"Time"* and *"Who"* columns represent the total encapsulation of the customer's management of temporal (day-timer) and personal (phonebook) contextual information as *capabilities* independent of each other and of the services to which they might be applied.

The service context profile depends upon these capabilities only in an *abstract symbolic* manner that does not depend upon their particular implementations. The encoding of temporal information by the day-timer, etc. is isolated from the remainder of the system. For example, the management of group/individual inheritance within the customer's phonebook, and the combining of states within the customer's state manager are wholly contained within their respective implementations. The substitution of a new day-timer with a new encoding of *"WorkHrs"* will not affect the *symbolic meaning* of *"WorkHrs"* within the service context profile, or its real-time processing by the customer's feature manager.

Consequently, application services and features no longer are developed as comprehensive self-contained functions which individually incorporate proprietary day-timers, phonebooks, etc. Such service-generic functionality's as a day-timer, a phonebook, and a customer state manager are no longer considered to be embeddable components of application functionality's. This approach to service design and development does require rethinking by the TINA community as to what should constitute a service or feature. The following material will hopefully contribute to this re-evaluation.

The customer no longer must subscribe to monolithic service packages—a la, Wide Area Centrex with PIM override—in which each component functionality is tightly integrated (embedded) with the others. Instead, the customer now is able to mix and match various feature capabilities as he so wishes. The customer performs the final integration of those subscribed capabilities—via customer-centered service management—to achieve the personalized behaviors which are desired.

Efforts have been made by various individuals and standards bodies—as well as by TINA—to identify standard sets of *service independent building blocks* (**SIB**'s) from which services and features could be tightly integrated—to create monolithic service packages. This quest for logical SIB's still is important, but a new approach to SIB integration is required. This new approach to SIB development should produce *capabilities* which are—*hot*, i.e., real-time—*plug-n-play* in a customizable service management sense, rather than, integrable in a service creation (packaging) sense.

For each service-generic functionality, e.g., the day-timer, a standardized set of interface methods are defined which all implementations of that functionality support. For example, each implementation of (a given level of) the day-timer functionality must support querying to determine which, if any, of its time entities are active at a given moment in time, to determine where two given time entities intersect, to determine whether one entity is dependent upon—built upon, inherits from—another, etc.

3.5 Symbolic Declarative Information

Information management and feature processing are partitioned horizontally *into layers of increasing abstraction and generalization. The customer is provided with the ability to state his problems (desired service behaviors) and their solutions—in customer-understandable terms—in as abstract and as general a manner as possible.*

For each layer of functional abstraction, the processing of the other layers with which that layer may interact are *symbolically* represented. The purpose for this symbolic representation is to enable each given layer not only to invoke—*procedurally*—those other functionality's, but also to provide the basis for that layer to *reason* regarding the benefits, costs, etc. of each such invocation. The provision for such a *reasoning* capability by each system about its alternatives and their consequences is fundamental to the development of truly *intelligent* networks.

As a general concept, knowledge may be encoded *declaratively* in passive data structures which are interpreted by other procedures, or *procedurally* as programmable units that encapsulate that knowledge. As a comparison with traditional programming concepts, declarative knowledge is analogous to the source code of a program, while the procedural knowledge is analogous to the object code of a program.

Not only may source code be executed interpretatively, but also (incrementally) compiled for more efficient execution. In a procedural sense, it is self-documenting—to anyone or thing able to interpret it. A given block of source code may be analyzed from a number of viewpoints. For example, it's resource utilization, solution approach, etc. can be compared with those of alternate source code which might satisfy the same computational need more quickly, efficiently, etc.

Such is the potential power of a symbolic declarative representation of the information that comprises the customer's service administration. The information that the customer provides has the potential to be utilized in many contexts and for many purposes beyond its original intent, as is shown in later sections.

As examples of a symbolic declarative representation, the service contexts of the service context profile each *symbolically* associate various types of contextual information with the execution of the customer's subscribed service features. Not only the customer's contextual entities but also the system behaviors of the subscribed features that are moderated by them are represented in the service context profile in implementation-independent symbolic form! Each service context is a symbolic statement about service behavior or activity desired by the customer, independent of any representations and implementations of the underlying entities which these symbols represent.

Consider the following entry from Table 1 in Section 2:

Late	*	OnCall	Busy	FCW	Alert	Loud

The time period *"Late,"* the customer state *"OnCall,"* the system state *"Busy,"* the service feature *"FCW"*—flexible call waiting, its parameter option *"Alert,"* and its value of *"Loud"* are all—customer or system-defined—*symbolic labels* which are used by the customer to express a desired service behavior along with the circumstances under which that behavior would be desired.

The *symbolic interpretation* of the meaning of this service context is independent of the current implementations of the entities which these symbols represent. As the network evolves through future AIN call-model releases, etc., these implementations may change. However, the *symbolic meaning* of this service context statement of the customer remains valid, meaning the same conceptually as it did when originally expressed.

In general, the customer may be permitted to define hierarchies—or even lattices—of successively complex time periods, groups of individuals, sets of abstract customer states, etc. Identifiable system states and service feature capabilities, likewise, are symbolically characterized (classified, modeled) within the AIN *service description profile*. Their corresponding implementations are managed within the *service management profile*. The typical residential customer might view any cause for a busy state in the AIN as simply *"Busy."* On the other hand, a business customer may wish to distinguish between line-side and trunk-side busy, etc.

The service contexts portrayed in Table 1 as rows of a spreadsheet are more accurately represented as declarative rules of a rule-set, as suggested by the last example in Section 3.2. The customer symbolically expresses the conditions and characteristics of the service behaviors desired. The domain of *predicates* (or conditions) and of *actions* (or behaviors) the customer may express is governed by the service feature capabilities to which he has subscribed.

3.6 Flexible Processing Strategy

The symbolic declarative representation of the customer's service management information is neutral to any specific service feature. This provides the basis for a virtually unlimited flexibility as to what applications such information may be applied, and as to how and when that information may be utilized..

Consider the task of selecting from a customer's service context profile a context rule that is *applicable* to process at a given moment in a call's execution. One source of information—e.g., temporal conditions from the day-timer—could be examined, and the results then used to limit or focus consideration of other sources. Which source should be processed first? the customer's day-timer? her phonebook? her state management? the system's state management? ... Could multiple sources be examined in parallel and their results then compared?

Many strategies could be implemented to support real-time processing of this information. On the one extreme are implementations that for reasons of performance would encode this information rigidly into procedural decision graphs with process flows programmed explicitly as to when and to how this information is used. At the other extreme are quite flexible open-ended production system approaches such as found in many AI (artificial intelligence) knowledge-based systems.

The declarative information model of this architecture is sufficiently flexible that the information in the customer context and service context profiles indeed can be *compiled* to generate optimized but rigid procedural decision graphs—such as might be generated for current AIN SCP's—based upon a specific procedural processing strategy. Thus, the traditional procedural approach is readily implementable from this declarative information model.

However, all possibilities (not only those that are normally expected, but also exceptional cases) need to be addressed by such a procedural version. In the case of one commercial AIN SCP, the decision graphs (called CPR's, i.e., customer provisioning records) typically contain as much or more logic for exception handling than for normal processing.

Any changes to the information in the profiles or to the established processing procedures would require regeneration of the decision graphs. This approach is analogous to the rule-set compiler technology used by some expert system shells which convert the declarative knowledge base into a procedural program—effectively, a *decision graph*. Fortunately, compilation to a procedural form is not the only way to achieve the desired real-time performance.

The strategic approach—more flexible, extendible, scaleable, etc.—is to employ representations that are enhanced for real-time processing but do not sacrifice of the declarative nature of the information. The flexibility with which the declarative form of information may be applied during real-time more than compensates for apparent performance gains that rigid procedural compilation might produce.

Generally, the service management representation of information must accommodate the customer's view and support efficient interactive editing by the customer; while, the real-time representation needs to be optimized for query processing. This is a generalization of the problem that every DBMS must address. In fact, analogues of DBMS solutions are applicable here.

The strategic is to use an optimization approach which compiles the *representation*, but not the *information*! As an example of such, recall in Section 3.2 that the *Label* concept was introduced as the *symbolic layer* of abstraction between any customer interfaces that might be implemented, and any AIN applications that might use the *information* entered via those interfaces.

By its design, the Label abstracts (compiles) the customer interface details—e.g., particular language, customer-centered names, device—from the information that the real-time SCP of the AIN needs to process the customer's features, since the real-time processing of that information is not dependent upon such attributes. On the other hand, such details still are available via the Label mechanism should they be needed, e.g., in support of an IP-related feature.

The author has described elsewhere a day-timer concept which utilizes a customer-centered semantic-net-like implementation by which the customer expresses temporal entities as well as a bit-packed encoded form of that information for real-time use. All the temporal *information* content of the one is present and accessible in the other!

Yet another approach to the *conditioning* of declarative information for efficient real-time processing is analogous to the hash functions, etc. that are used by relational databases to improve their performance. The result of this approach is a real-time implementation whose performance should approximate that of the compiled decision tree, but which still provides the greater flexibility of the declarative form. *Declarative hashing* can be implemented more intelligently than the arbitrary hash mechanisms used by relational databases.

The proposed AIN architecture implements the conditioning approach as a *feature management context*. Based upon its analysis of the customer's other context profiles, the AM maintains the *feature management context* to provide guidance to the real-time FM as it focuses processing upon temporal, personal, state information, etc. in whatever strategy might yield better performance in the selection of the next *appropriate* service context entry to be considered.

This analysis can utilize *feedback* from the customer's prior service usage—such as would be available from the billing system. Via such analysis, the AM (predictively) predisposes the FM via the *feature management context* to adopt (consider first) certain processing strategies over others. Any valid strategies still remains available to the FM, should its use be indicated by other real-time events.

The AIN thus has the capability to *adapt* to (to *learn*) those feature behaviors that appear to be more characteristic of the customer. The net effect of this conditioning approach is that the representation and the organization of the information—and of the systems which interpret and process it—are subject to continual tuning. The consequence is that a new *logical* decision graph is continually being updated based upon new information from the customer or feedback from the system.

3.7 Information Abstraction & Late-Binding Interpretation

The declarative symbolic information of one layer represents an abstraction (model) of the processing (procedures) of lower functional layers. Under different real-time circumstances, the same information abstractions may be mapped to different realizations, and may be applied in different ways to satisfy quite different requirements. Information abstraction leads to information reuse.

The *interpretation* of symbolic phrases such as *"Alert Loud"* and *"Alert Quiet"*—from the service context profile of Table 1 in Section 2—is dependent upon the real-time contexts in which they are applied, as well as upon customer service specifications and preferences. These symbolically expressed feature behaviors could be interpreted to mean to *"buzz"* versus to *"vibrate"* the customer's pager in one instance, and in another instance to mean to display a *"large"* phone icon in the center of customer's PC screen, versus a small blinking icon in its corner in another instance. These interpretations also are based upon the results of the feature negotiation process explained in the next section

This customer-centered information abstraction focuses upon the customer's *problems*—e.g., the need for alerting and security—rather than upon currently available AIN *solutions*—e.g., the use of paging, distinctive ringing, PIN's, and voice prints. This abstract symbolic information that today is applied to the operation of a pager or the invocation of a distinctive ring, could just as easily be applied tomorrow to a device, mechanism, or process not even conceived of today—but which provides a solution to the customer's problem! The customer's problems tend to be persistent; while, current AIN solutions to those problems can be transitory.

This approach to AIN service delivery is analogous to the mathematician who has expressed (modeled) a physically meaningful problem (electrical, structural, etc.) as a set of equations (algebraic, differential, etc.), whose solution (if solvable) provides the solution to the problem. The statement of the problem as a set of symbolic equations represents an abstract solution of the problem. As new methods (e.g., numerical analysis) and new mechanisms (e.g., computers) become available, they readily may be applied to this general problem statement to expand the size, reduce the time, increase the accuracy, etc. of any specific solution.

The concept of *late-binding* (or *lazy-binding*) refers to postponement of the determination of the *binding* (association, matching) of an object, symbol, etc. with its value, interpretation, processing, etc. until that binding is actually needed.

The information contained in the various context profiles consists of *symbolic patterns* whose meanings are (directly) determined, or (indirectly) inferred, etc. as the result of appropriately *pattern-matching* their information abstractions with potential realizations and interpretations. Examples of the former case would be the resolution of *"WorkHrs"* or *"MyFamily"* from the customer's PIM. The resolution of appropriate meanings (realizations) for *"Alert Loud"* and *"Alert Quiet"* are examples of the latter case.

As described in Section 1.2, the *service description profile* provides a comprehensive declarative view of AIN system features and capabilities and how they could be applicable to various categories of customer problems. For example, distinctive ring and paging are identified as capabilities that can be used to provide alerting.

As described in Section 1.3, the *service management profile* associates these declaratively described features and capabilities with details regarding their implementations within the AIN. This associated information may identify specific platforms (e.g., SSP, IP) where they reside, specific resources they require—e.g., access to a customer's voice-prints, access to a Radish-capable CPE, etc.

One critical characteristic of intelligent entities is their *self-awareness*. Elements of this characteristic are beginning to appear in today's computer systems in the form of *plug-n-play* hardware, and *ORB-like* and *OLE-like* software. As part of their implementations, these components have such self-knowledge as to how they should be configured, to what capabilities they can provide, etc.

This *possession of self-knowledge* is characteristic of the features and capabilities available in this AIN architecture. Not only can an AIN capability provide its designed functionality when requested, information also can be provided the potential requester regarding what it does, how and when it is done, what resources are required, etc. In other words, each AIN capability is self-explanatory sufficiently for the potential user to determine its applicability to the potential user's need.

The achievement (representation) of this self-knowledge is another example of the application of symbolic declarative information. The symbolic declarative information of one functional layer possesses an abstraction (model) of the processing (procedures) of lower functional layers. This concept of self-knowledge is applied recursively from a capability's highest level of functional abstraction—e.g., *"Alert Loud"*—to the lower-level objects, functions, etc.—e.g., the TCAP messages and parameters—which implement it.

Within this architecture, an *information duality* exists. From any level of abstraction downward, there is both the recursive interpretive self-knowledge form of that functionality, as well as the compiled—no self-knowledge, procedural-like—form of that functionality. This concept may be viewed as a generalization of the approach taken by a numeric application that provides a software-based implementation of floating-point arithmetic for use in circumstances where a hardware-based floating-point unit is not available. Another analogous example of this type of duality is the normal kernel of an OS from which internal symbols have been *stripped*, and the *debug kernel* in which they have been retained to assist software developers.

Under normal circumstances when the customer's services can be delivered as requested, the self-knowledge of a capability may not be needed at real-time by the FM. The capabilities identified by the customer's AM have been *matched* to the requirements of the customer's service contexts. The customer's FM needs only to invoke those capabilities as previously determined by the AM.

On the other hand, this self-knowledge is crucial to the feature interaction and negotiation management described in the next section. The recursive descent into the self-knowledge of a feature capability by the customer's AM needs be only deep enough to satisfy the AM as to the appropriateness of a capability to satisfy the contexts of the customer's service profiles. The FM may be confronted with intelligently improvising a workable solution for the customer. This self-knowledge is crucial to that task.

3.8 Feature Interaction & Negotiation Management

The customer's profiles are treated by this AIN architecture as representing the customer's preferred service feature behavior, rather than as the expression of a set of rigid specifications. The goal of the FM is to select the service contexts which better satisfy the customers preferences and also are compatible with real-time system states and capabilities. In particular, the FM's of interacting customers utilize these symbolically expressed preferences as the basis for negotiation of service behavior that is acceptable to those parties.

The identification of *possible* interactions, and the detection of *real* interactions in AIN architectures where each service has been designed and implemented to execute independently of all others is a nightmare situation. Much of feature interaction conflict is associated with the ambiguous application of multiple services to the same context. As an example, in the case of *single-customer* feature interaction—i.e., the services of one customer, flexible call forwarding (FWF) would forward the caller while flexible call waiting (FWD) would place the caller on hold. To require the customer to choose between subscriptions to these two services is a rather extreme approach to feature interaction management, as is to arbitrarily prefer one over the other.

An example of *multi-customer* feature interaction exists when a customer being called desires to perform call screening, etc. based upon calling name delivery (CND); while, the calling customer desires to block the delivery of his CND. Such feature interactions are not so easily resolved. Either one or both customers fail to receive benefit of the services to which he subscribed. The complexity of possible feature interactions will be too diverse and sophisticated for system managers and service developers—independent of the customer—to provide general arbitrary solutions that satisfy a majority of customers, let alone are satisfactory to everyone.

A method by which *appropriate* features for each possible context may easily be specified by the customer would greatly reduce the magnitude of the feature interaction problem. Features that supposedly *conflict* could be *coordinated* to exhibit the opposite appearance of *cooperation* in the view of the customer—which is the opinion that really matters! Any approach to feature interaction management must be a customer-centered approach to be successful.

To facilitate such an approach in the case of *single-customer* feature management, this architecture permits the customer to organize his contextual information in a manner which is consistent across all services, service management systems, etc.—as has been explained. The *feature management process* is decoupled from any particular application service into an AM that manages the resolution of the customer contextual information, and an FM that manages the execution of his services, based on the results of that resolution.

In the case of *multi-customer* feature interaction between the feature behaviors of different customers, *feature negotiation management* provided by the AIN negotiates a *fall-back* (or alternative) position to service delivery, so as to avoid feature interaction *deadlock,* while providing the involved customers with an *acceptable* (though perhaps less than optimal) level of (or alternative to) the original service behavior.

This negotiation concept is applicable to *mobility-oriented* services such as UPT (universal personal telephone) in which the remote UPT location may not be able to support the full set of customer services as subscribed. The management of the services of multiple customers, each of which is *registered* to a common CPE, is amenable to this approach.

The principles of negotiation are generally understood by everyone. *Negotiation* is an iterative process of *announcement-bid-award* in which an entity poses its need to other entities who might be able to satisfy the request, receives offers from responders, and accepts the best offer of assistance. Variations in the negotiation process include the possibility of iterative offers and counter-offers, of teaming arrangements, etc.

In each instance of negotiation, three fundamental components may be identified:

- *exchange of information* -- by requests, responses, etc.,
- *evaluation of exchanged information* -- by each entity from its own local perspective, and
- *final agreement* (contract) -- by mutual selection.

In preparation for a solution by negotiation, there are *preparatory activities* which each involved party should complete. These include such items as the clarification (from each negotiator's perspective) of:

- *absolute bounds* of this give-n-take effort,
- *compromise fall-back positions*, multiple, if possible, and
- *evaluation criteria* by which to judge any and all offers.

In support of the negotiation process, mechanisms such as languages (data structures), and protocols (negotiation rules) must be in place by which each party may *accurately* and *succinctly* communicate requests, bids, etc. Under the assumption that the involved parties are willing to cooperate, negotiation provides a reasonable vehicle for the identification of *common ground,* i.e., a *global* view of the problem's solution which is *locally* acceptable to each party involved.

In general, the system's negotiation mechanisms should facilitate the customer's indication of *fall-back* service behaviors which he would accept. For example, the customer may prefer ADSI prompts, but would accept voice prompts, or even distinctive tones, were ADSI prompts not available. Much of multi-customer feature management is the matter of determination of an acceptable level of service delivery *common* to the involved parties, similar to the functionality now provided by *smart* modems that support multiple bandwidths, error-correction and data compression schemes, etc.

The information in the customer context and service context profiles is treated by this AIN architecture as representing the customer's *preferred* or *approximate* service feature wishes or desires, rather than as the expression of a set of rigid specifications. While the AIN always will seek to satisfy the customer's requested feature behavior exactly as specified, the AIN is permitted (empowered) to *satisfy* these requests as well as it can.

So long as the FM can deliver the customer's services as requested in the service context profile, the customer should not experience any undesirable *single-customer* feature interactions. Circumstances can arise, however, in which the customer cannot be provided his services as requested. This situation can be due to *multi-customer* feature interactions—if another customer's wishes conflicts with his—or *inter-network* feature interactions—if the public network or CPE technically is not able to provide the services as requested.

Under such circumstances, the customer's FM provides *feature interaction negotiation management* in behalf of the customer. The goal of this process is to provide the customer with alternate service behavior that is as *satisfactory* as possible, while avoiding behaviors that definitely would be *unsatisfactory*. What may be called *satisfactory* or *unsatisfactory* is determined from the (explicit or implicit) *alternatives* identified from the customer in his service subscription and service context profiles, or otherwise by system-provided defaults.

A solid foundation for *feature interaction negotiation management* is provided by the concepts and capabilities of this AIN architecture described in Sections 3.1 through 3.7.

The counterpart to the symbolic declarative constructs that are the basis of the customer's service administration is the incorporation of self-knowledge within AIN features and capabilities. Just as the customer can create and organize the entities of his administrative PIM and customer state management, the developers and management of the AIN provide this self-knowledge support as part of the development and deployment of each AIN feature and capability. This process may be viewed as a generalization of the process of associating keywords with a technical paper to assist potential readers in their search for appropriate reference materials.

Just as the IEEE provides a taxonomy of codes by which a member can classify himself professionally, so also are implemented symbolic declarative taxonomies of customer requirements—e.g., alerting, privacy, security—and of AIN capabilities—e.g., distinctive ringing, call-blocking, PIN's—which are cross-referenced. These self-knowledge taxonomies range from quite abstract to quite specific. *"Alert," "Alert Quietly," "Alert Quietly w/ Pager,"* and *"Alert Quietly w/ Pager LCD only"* are examples of four levels of customer alerting of increasing specificity. With *"Messaging"* considered to be a weak form of *"Alerting,"* passive rather than active, the leaving of a message is one way to leave an alert (notification) that the customer would be expected to receive eventually.

The following is a listing of *feature negotiation management heuristics* (i.e., *rules-of-thumb),* that define *strategies* which the customer's FM may apply as it attempts to satisfy the customer with acceptable feature behavior:

- Satisfy the request at a level of *specificity equal to or greater than that of the request*.
- Satisfy the request based on a level *less specific (more abstract) than that of the request*.
- Partially satisfy the request based on a *subset of the requested capability*, if such exists.
- Partially satisfy the request based on an *economical* (e.g., monetary-wise, resource-wise) *generalization of the requested capability*.
- Partially satisfy the request based on any *generalization of the requested capability*.
- Alternatively satisfy the request based on a *generalization of the service context*, e.g., by weakening the customer's constraints to identify another less specific service context with an associated capability behavior that could be satisfied by one of the above heuristics.
- In a *backward-chaining* sense, is there identifiable another service context that could be accomplished, which could make the currently focused service context decidable?

These heuristics are representable as pattern-matching problems that may be implemented using production system (rule-based) technology. Each heuristic may be implemented as a *knowledge source* of a *blackboard*-like distributed artificial intelligence system. The customer's FM provides the role of the blackboard monitor. Again, the symbolic declarative nature of this AIN architecture is crucial to the realization of such feature negotiation.

Of particular interest is the second heuristic above that relates to the solution of a problem at a more abstract level than it is originally stated. Consider the case where a customer is provided service by one AIN service provider, but is currently located (registered) physically in the jurisdiction of another provider—similar to the *roaming* problem in cellular service. Suppose that the AIN SCP's of the two providers supported communication of the customer's FM at his service provider's SCP with the FM that has jurisdiction at the customer's remote (roaming) location.

These two FM's via feature negotiation are able to *virtualize* the customer's services at the remote site. The *"Alert's,"* etc. are mapped (negotiated) to acceptable behaviors at the guest location. Likewise, low-level system states at the guest location are abstracted and passed to the customer's SCP where they are used to match service contexts, from which behavior requests flow back to the guest location's SCP.

4. Conclusion and Remarks

A high-level view of an information model and system architecture for the AIN—the Advanced Intelligent Network—has been presented which places the customer as the focus of all service design, development, deployment, and management. In particular, this paper has indicated how several technologies from the general field of distributed artificial intelligence (DAI) may be applied with far-reaching consequences.

The presentation has been broad and high-level so that the continuity of all the presented concepts could be emphasized. The references below contain more detailed information regarding how these concepts may be implemented.

Some of the ideas promoted here will require near-radical change to the approach now being taken by the industry regarding all—not restricted to AIN—service design, development, deployment, and management. The realization of all these concepts will necessarily be an incremental one. They have been presented in chronological order—what must be implemented first, second, ...

The quantity and variety of AIN services that have been developed under the traditional mode are relatively small and limited. Hence the AIN environment is a natural place to begin design and implementation of these concepts.

Currently, effort is being focused on the mapping of these concepts to currently available AIN platforms (e.g., AT&T's AI-Net, and Bellcore's ISCP). These concepts are also being applied to the upgrade and enhancement of our OAM&P. Everyone is affected from the product management organization that conceives of new services through to the operations support organizations that maintain the network on a daily basis.

Hopefully, an open dialogue will be generated around these concepts and result in a much more adaptive and user-friendly public network.

5. References

[1] Bach, Maurice J., The Design of the UNIX Operating System, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

[2] Bobrow, Daniel G. and Allan Collins, Editors, Representation and Understanding--Studies in Cognitive Science, Academic Press, New York, NY, 1975.

[3] Bond, Alan H. and Les Glasser, Editors, Readings in Distributed Artificial Intelligence, Morgan Kaufman Publishers, San Mateo, CA, 1988.

[4] Davis, Ernest, *"Constraint Propagation with Interval Labels,"* Artificial Intelligence, North-Holland Publishing, 32:281-332, 1987.

[5] Davis, Randall and Reid G. Smith, *"Negotiation as a Metaphor for Distributed Problem Solving,"* Artificial Intelligence, North-Holland Publishing, 20:63-109, 1983.

[6] Ellis, Gerald, *"Compiling Conceptual Graphs,"* IEEE Knowledge & Data Engineering, 7:68-81, 1995.

[7] Engelmore, Robert and Tony Morgan, Editors, Blackboard Systems, Addison-Wesley, Reading, MA, 1988.

[8] Forgy, C.L., *"Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem,"* Artificial Intelligence, North-Holland Publishing, 19:17-38, 1982.

[9] Graver, Jack E. and Mark E. Watkins, Combinatorics with Emphasis on the Theory of Graphs, Springer-Verlag, New York, NY, 1977.

[10] Griffeth, Nancy D. and Yow-Jian Lin, Eds., Special Issue: *"Telecommunications: How Many Features Can You Add?"* IEEE Computer, Aug. 93.

[11] Griffeth, Nancy D. and Hugo Velthuijsen, *"The Negotiating Agents Approach to Runtime Feature Interaction Resolution,"* Feature Interactions in Telecommunications Systems, IOS Press, Washington, D.C. pp.217-235, 1994.

[12] Lin ,Yow-Jian and Nancy D. Griffeth, Eds., Special Issue: *"Managing Feature Interactions in Telecommunications Systems,"* IEEE Communnications, Aug. 93.

[13] Miranker, Daniel P., TREAT: A New and Efficient Match Algorithm for AI Production Systems, Morgan Kaufman Publishers, San Mateo, CA, 1990.

[14] Mylopoulos, John and Michael L. Brodie, Editors, Readings in Artificial Intelligence & Databases, Morgan Kaufman Publishers, San Mateo, CA, 1988.

[15] Pearl, Judea, Heuristics--Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, MA, 1984.

[16] Smith, James T., *"AIN System Design & Information Model Proposal--A Data-Driven Approach,"* GTE Internal Document Jan. 94.

[17] Smith, James T., *"An Approach to Customer-Centered Interfaces,"* Third International Conference on Universal Personal Communications, pp. 619-623, Sept. 1994.

[18] Smith, James T., *"Meeting the Service Creation Challenge Using the ISCP,"* GTE Service Creation Workshop-94, Nov. 1994.

[19] Smith, James T., *"AIN System Development: The Customer Centered Service Context Prolile,"* International Communications Conference (ICC-95), Jun. 1995.

[20] Waterman, Donald A., A Guide to Expert Systems, Addison-Wesley, Reading, MA, 1986.