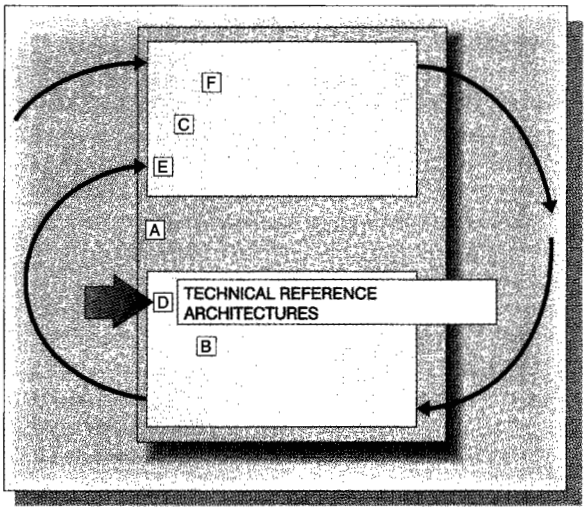
Technical reference architectures

by P. T. L. Lloyd
G. M. Galambos



Today's approaches to solution development are still primarily based on "handcrafting" and bear little relationship to the asset-based engineering methods so successfully used in other disciplines. In this paper we argue that these handcrafting approaches have passed their "sell-by" dates, and a more disciplined and constrained method of system development is needed. We focus particularly on the definition of technical architectures as the basis for constructing applications. We argue that a constrained set of reference architectures for a given set of problem domains is not only feasible, but mandatory for large-scale enterprise development, and we provide some example fragments of reference architectures for the administrative systems domain. These reference architectures and their successors, harvested and continually refreshed from successful consulting engagements, will form the basis of IBM's asset-based approach to solutions development in the future.

In this paper we focus on *enterprise-scale* solutions. When we refer to enterprises we are thinking of large businesses, typically with thousands of employees and, in many cases, thousands to millions of customers. The information technology (IT) systems being developed may be laid out over wide geographic areas, transaction rates may be tens or hundreds per second, and the company's corporate databases may be many thousands of gigabytes in size. Service to these companies forms the bedrock of IBM's (and many IT companies') business, and is key to IBM's future success.

Although we approach the subject from an IBM perspective, we believe that our comments apply widely to the IT industry in the context of enterprise-scale development.

An asset-based approach to solution development

Today's approaches to solution development are still primarily based on "handcrafting" and bear little relationship to the asset-based engineering methods so successfully used in other disciplines. We believe that handcrafting approaches are obsolete and a more disciplined and constrained method is needed for solution development.

Problems with the current approach. Today, the most common model for solution development is

based on an approach that we call "heroic." A highly skilled, energetic team studies and refines the stated requirements, defines the architecture for a solution to meet those requirements, carries out a detailed design, and builds the system. Assets brought to the table are primarily the skills and experience of the individuals who make up the team.

Size, complexity, and risk. Whereas in the 1970s and to some extent the 1980s the heroic approach seemed reasonably effective, it is clear that for enterprise-scale systems it is no longer adequate. In 1996 surveys of 20 major enterprise solution development projects in North America and Europe showed that solution development project risk increased rapidly with project size. For projects requiring more than 100 person years, the risk of project delay or failure was very high.

It is not surprising that enterprises are seeking to mitigate risk by taking an asset-based approach.

Moreover, for various reasons, the amount of project resources needed to replace existing business systems is rising rapidly. In the 1960s, utility billing systems could be developed in 50 person years using batch technology; in the 1970s, the first-generation on-line systems (which provided the genesis for IBM's CICS* [Customer Information Control System] transaction monitor) were needing 100 person years or so; in the 1980s, the figure was up to 150 person years with more sophisticated personal computer (PC)-based interfaces. However, by the mid-1990s, with the advent of object-oriented front ends and distributed logic client/server and other technologies, several projects were up to 500 person years with an elapsed time of four to five years. A considerable part of this effort, perhaps one-third, was spent on integrating new function with existing (legacy) IT systems and handling data migration to new database designs. These figures are not unique to the utilities industry—the same trends are apparent in the insurance industry. Specifically, the replacement of existing contract management systems in a large insurance company in France was recently reported as consuming 500 person years over an elapsed time of six years or more.

It goes without saying that any project of 500 person years is exposed to considerable risk—not only the risk of technical failure, but also the risk that upon final delivery, the business requirements will have changed beyond recognition and the delivered solution may no longer meet business needs. When we combine this risk with the likely development costs of $50 million or more, it is apparent that our industry is facing a crisis.

In these circumstances, it is not surprising that enterprises are seeking to mitigate risk by taking an asset-based approach. Whereas ten years ago, an enterprise would likely have seen advantages in undertaking a custom development, today the inclination would be to look for a suitable packaged solution and customize it as needed. In other words, in trying to gain business advantage, the focus has changed from the development of specific applications to the specific tailoring of generic applications.

"Silo" or "stovepipe" solutions. Older, or "legacy" systems frequently seem to the end user to be disconnected, with independently operating "silos."[1] For example, in an insurance company, one IT system may deal with life insurance business, and another with general business such as car insurance. More likely, the life insurance business itself will run multiple systems to handle various products such as unit business, group products, and so on. With today's focus on customer service, it is increasingly unacceptable for an enterprise to communicate with its customers separately from each system. Put more positively, a more coherent view of a customer that considers all products held by that customer provides excellent marketing opportunities and improved customer retention. So the existence of silos, manifested typically both through incompatible hardware and incompatible or segregated software, is a serious issue.

Many solutions developed in the last five years have unfortunately become "instant legacy" solutions. Rather than solving the silo problem, they have contributed to it, through inappropriate distribution of data or function to fat clients,[2] or incompatible middleware or databases, for example.

So both clients and vendors, such as service providers or packaged solution suppliers, need some kind

of mechanism to stop the growth of silos, which are almost as damaging to the vendors as to the enterprise clients they serve. We view an asset-based approach to architecture as a primary mechanism to address this problem.

Lack of a standard programming model or strategy. In the early 1980s, for the enterprise-scale administrative systems that are the focus of this paper, most large customers had settled on an architecture involving IBM 3270 displays linked by wide area network to an IBM System/370* or IBM-compatible mainframe running MVS (Multiple Virtual Storage), CICS, and a relational database management system (DBMS) from IBM or one of the other large DBMS vendors. The programming model was basically CICS and SQL (Structured Query Language).

Today, there is no such agreement on a programming model across the large enterprise IT departments. Fat-client approaches have had some success, but rarely at enterprise scale, and consulting organizations such as Gartner Group[3] are recommending a more sophisticated model with distributed logic and multiple tiers. Unfortunately, there is as yet no pervasive, standard approach to constructing such systems. As a result, development projects are frequently creating "one-off" middleware to hide complexity from the application developers. Such middleware can consume hundreds of person years and (as importantly) can cause unpredictable delays in the project schedule.

The world has changed a great deal since the early 1980s, when the IBM strategy for IT development was an essential part of every enterprise IT vice president's knowledge set. The advent of PCs, TCP/IP (Transmission Control Protocol/Internet Protocol) and the Internet, object-oriented technology, and open systems has set the IT industry free, but at what cost? Today, each enterprise is free to define its own strategy—but this is proving a costly form of freedom and could be considered a distraction from the IT department's real goal of providing cost-effective solutions to business problems and opportunities.

Jacobson et al.[4] state that first-generation attempts at software component reuse have largely failed. Many companies, including IBM, have tried to establish a practice of reuse without the necessary processes and disciplines to provide the required support, and in too many cases disillusionment has set in. Based on considerable experience, Jacobson et al. state that architecture is a key to reuse. This is also supported by Bass et al.[5]

The "heroic" model of development We mentioned earlier the heroic style, characterized by skilled, enthusiastic development teams employing no systematic reuse of assets other than their own experience, and some of the problems it has faced. There are other problems involved with this approach.

First, it is extremely difficult to estimate the cost of development in such an environment. Because there is no yardstick (other than the personal experiences of the team) to assess the scope of the proposed solution, the scope may well be incorrectly assessed, leading to estimating errors in the cost of development, and "scope creep" later, as the true impact of the requirement is recognized. Every project becomes an adventure, with middleware invented as the project proceeds.

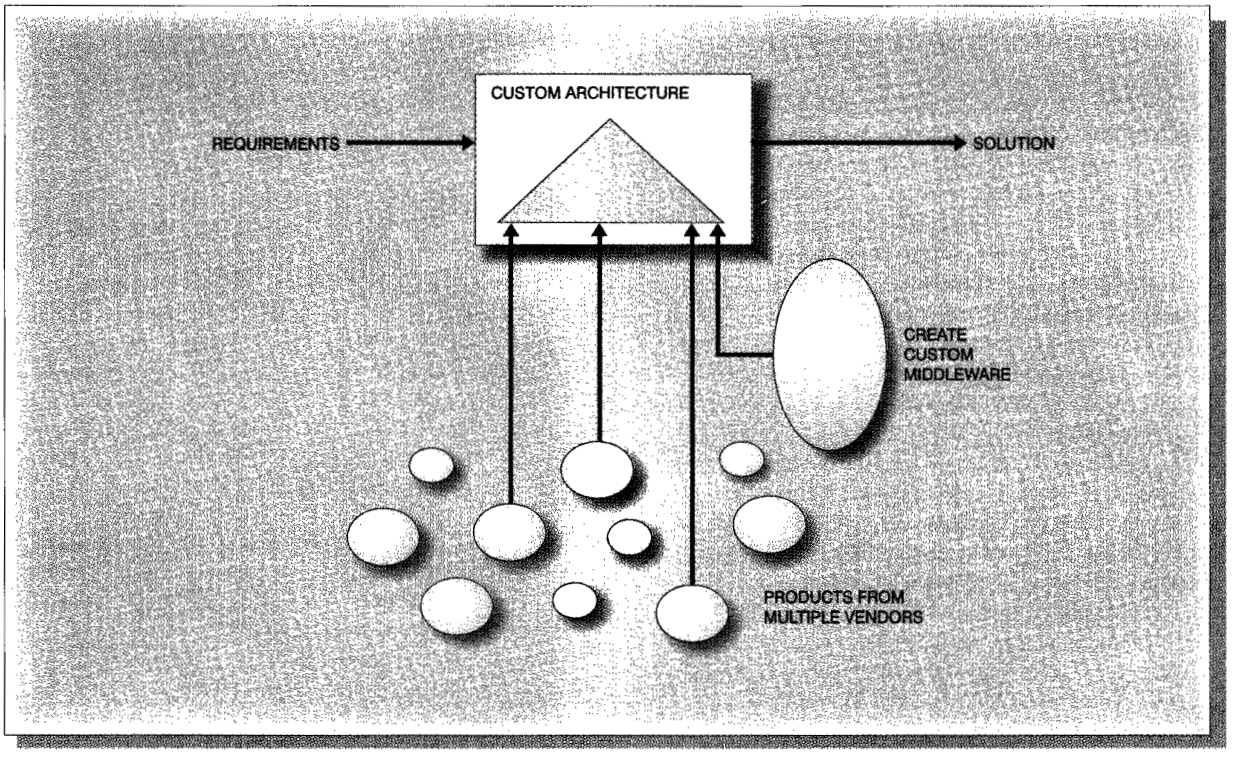
Second, with no pervasive standards for methods, work products, and other assets, it is very difficult to assemble teams efficiently. New staff need to be trained on the particular methods used on an engagement, and "crash" training is unlikely to be fully effective.

Third, there are typically no rigorous standards for documentation of the finished system that would allow easy reuse of it or its components. So there is no easy way to improve next time around. The heroic approach is illustrated in Figure 1.

Development starts "from scratch," and architectural components are integrated to create a custom architecture without use of a template or blueprint. Frequently, the project creates extensive custom middleware, which might not have been needed if a blueprint-driven approach had been taken. The selection of products from vendors is left to the skill of one or a few key architects. As a consequence, the solution takes longer to develop and deliver, and the costs and risks are higher. Along the way, the opportunity to leverage existing products, services, and prior integration of components, and the opportunity to acquire knowledge for subsequent reuse have been lost.

The ESS asset-based approach. In mid-1996 IBM's Global Industries business unit, which has the mission of developing and supporting packaged industry solutions, set up the Enterprise Solutions Structure (ESS) initiative as a way to bring an improved,

Figure 1 The heroic approach to solution development can be slow, expensive, and risky.



asset-based approach to its solution development. The objectives of ESS were stated in an internal document as follows:

> The Enterprise Solutions Structure (ESS) establishes a standard architectural framework for IBM architects and designers to construct enterprise solutions based on the reuse of proven architecturally conformant assets. Specifically, this architecture provides a common, consistent approach for understanding and documenting business requirements via a business model, designing a logical architecture of key components and services, and finally, implementing a physical architecture based on actual products, platforms, and services.
>
> . . . For technical architecture, ESS establishes a specific set of predefined reference architectures . . . with conformant components and programming models that isolate business systems and applications from underlying operating systems, languages, and hardware platforms. The ESS components are cataloged in a consistent, extensible framework that encourages their use at the appropriate time in the design process.

The background to the ESS project is described by Plachy and Hausler.[6] In this paper we focus on the part of ESS that covers the selection and construction of technical reference architectures.

The role of architecture. The following quotation from Barry Boehm states the role of architecture well: "If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process."[7] We define *system architecture* as the structure or structures of the system, which comprise software and hardware components, the externally visible properties of those components, and the relationships among them.

This definition is similar to that for software architecture found in the works of Bass et al.[5] and Shaw

and Garlan,[8] but adds hardware components and the relationships between hardware and software to the definition. We believe that this is critical for successful development and deployment of IT solutions, for reasons that will be explored. We define a *component* as a modular unit of functionality, accessed through one or more interfaces. Components normally represent software (including operating systems), but can also represent firmware (e.g., PC BIOS [Basic Input/Output System]) or hardware (e.g., encryption devices, or interactive voice response units).[9]

Our focus is on the development of IT solutions for enterprises. A sound architecture is an essential prerequisite for the successful development of an enterprise-scale IT solution. Bass et al.[5] list three key reasons why this is so:

1. It aids communication among stakeholders. The architecture is a basis for ensuring mutual understanding.
2. It represents the earliest design decisions about a system. These decisions are absolutely critical in terms of development, deployment, and future maintenance cost. For example, in development they play a crucial role in team organization, and this reflects back later into the organization of the company for which the solution is being developed. In maintenance, the flexibility qualities built into the architecture will be key to the ability of the solution to respond to business and technology change.
3. It provides a compact and understandable abstraction of a system and hence is a mandatory prerequisite for the reuse of software assets. (See also Jacobson et al.[4])

At the beginning of a specific solution development project, the architecture team has the option of creating the architecture in a number of ways. It could develop the architecture "from scratch," or it could pick from a library of architectural styles, or it could reuse and build upon a reference architecture.

We define these terms as follows:

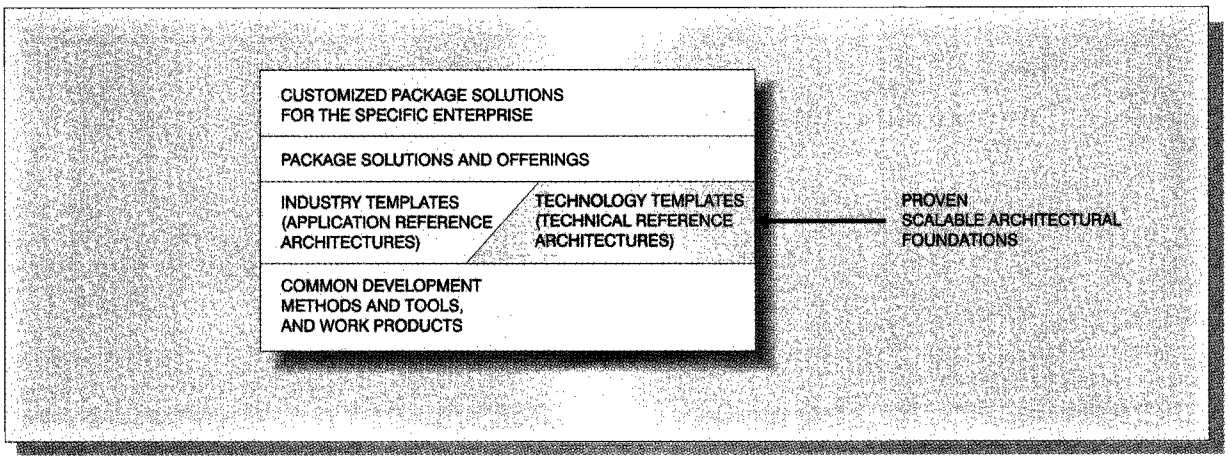
An *architectural style*[5,8,10] (or architectural pattern) is an idiomatic pattern of system structure, expressed as a description of component types and a pattern of their run-time control and data transfer. Examples include well-known styles such as layering, pipe-and-filter, hub-and-spoke, client/server, and topology patterns such as three-tier. These of course form part of the toolkit of every experienced IT architect. They are the architectural analogue of the design patterns used by experienced object designers,[11] building on the pioneering concepts of Alexander et al.[12] in the field of structural architecture. Like design patterns, architectural styles provide a convenient and compact way to describe complex structures. They represent real architectural experience, but are not yet codified as precisely as design patterns—for example, the link between architectural styles and the qualities they support (such as performance, security, reusability) is usually implicit rather than explicit.

A *reference architecture*[8] is an architecture that has already been created for a particular area of interest. It typically includes many different architectural styles, applied in different parts of its structure.

A *technical reference architecture* is a type of reference architecture that does not directly include structures of application (business) behavior. In other words, it can be used as a base architecture or template for several different application types. It nevertheless still applies only to a specific technical domain. For example, technical reference architectures or fragments of technical reference architectures exist today in the domains of distributed object systems (e.g., CORBA** [Common Object Request Broker Architecture**]), compiler development, and Internet (Web browsers and servers). We will see other examples later in this paper for the domain of administrative systems.

In our definition, a reference architecture includes both *functional* and *operational* aspects of an IT system. The functional aspect is concerned with the functionality of collaborating software components; the operational aspect is concerned with the distribution of components across the organization's geography, in order to achieve the required service level characteristics.[9] A reference architecture is not only a software architecture; it also provides predefined structures for the placement of software on hardware nodes, and structures for hardware connectivity. It is our experience that this linkage is key to keeping the architecture "honest" and grounded. The effects of distributing software components across a network are far-reaching, and the network configuration must be allowed to influence the software architecture (for example, in terms of component granularity) if enterprise-scale solution development is to be successful.

Figure 2 Proven, reusable architectural patterns are a key step in the evolution of asset-based solution building.



The technical reference architecture supports and constrains the *application reference architecture*. The latter (business- and industry-specific) part of the architecture uses the structures provided by the technical reference architecture, which establishes (part of) the programming model for the application: the set of application programming interfaces that define all services that a middleware component offers to an application.

The contribution of technical reference architectures. During 1996 and 1997, a subproject of ESS carried out project surveys to check the feasibility of the technical reference architecture approach, identified and prioritized the technical reference architectures to be harvested, and set about the population of these reference architectures. A key principle was to avoid theoretical solutions, avoid invention, and instead concentrate on harvesting "best-of-breed" architectures or fragments from enterprise-scale projects that had demonstrated success.

In late 1997, additional sponsorship was obtained from IBM's Global Services business unit, which had independently concluded that IBM's customers, and IBM's service professionals worldwide, would achieve great benefit from an asset-based approach to solution development. The proposed business model can be expressed in a layered structure, as shown in Figure 2.

The technical reference architectures are seen as a set of technology templates, on which solutions (either specific solutions being developed in an engagement for a particular customer, or packaged solutions being developed by IBM Global Industries for sale to multiple customers) are based. The description of reference architectures is based on a common set of concepts (an "architecture description language" or ADL), shared between all the professionals involved.

In the next section, we describe the specific technical reference architectures that have been harvested, and review some of the challenges and experiences we met as this work proceeded.

ESS technical reference architectures

The ESS technical architecture team included senior architects representing the requirements of three industries—banking, finance, and securities; insurance; and utilities. These industry representatives provided the following guidelines:

- Concentrate on enterprise-scale (rather than departmental) solutions. The team believed that many of today's small solutions were not adequate for large-scale, mission-critical environments.
- Focus on "administrative systems" applications in the chosen industry segments. This would include most customer service applications, but exclude, for example, real-time control applications.
- E-business, call center, and business intelligence solutions should get particular attention. All in-

dustry segments saw great interest in these subject areas.
- Concentrate on the following technologies and computing styles: Internet access, transactional processing, collaborative processing, and mobile technologies
- Pay particular attention to integration with legacy systems, and avoid further development of silo solutions

We carried out architectural surveys of around 20 enterprise-scale solution development projects from North America and Europe. These projects were using a mix of technologies, from fully object-oriented, through object-based, to procedural, and included examples (some successful, others less so) of implementations of most of the subject areas and technologies recommended by the technical architecture team's industry representatives. Most of these projects used a heroic style of development, but we examined some projects in which an asset-based approach had been used.

At the same time we carried out a survey of the architectural approaches used by other vendors (for example, SAP AG and Forté Systems, Inc.) and key architecture developments within IBM (including work in progress on San Francisco*[13] and on Component Broker Series[14]).

Although the languages used to describe these architectures differed greatly, we were able to discover large areas of commonality in approach. For example, in transactional processing we saw a general move away from fat clients toward a three-tier, distributed-logic model, which was designed to answer the key requirements of enterprises for "single view of the customer" (i.e., silo avoidance), legacy integration, and improved systems management.

We concluded that it was feasible for IBM to define a relatively small number of technical reference architectures, based on harvesting the best practices from successful implementations. The harvesting process itself was not straightforward, and we foresaw that there would be particular challenges in keeping these reference architectures up to date as technology advanced. Key to meeting these challenges was to harvest from *all* appropriate engagements; then, the scale and scope of IBM's services and solutions businesses would keep the architectural asset base refreshed and at the forefront of industry practice. This process is illustrated in Figure 3.

In this approach, the appropriate technical reference architecture is selected from a constrained set, provided it is a reasonable fit to the requirement. The models are then customized, but because there is a standard approach, middleware designs and product code will be reused from previous engagements or industry sources. Only if a design or a product does not already exist to meet the required customized architecture will a new component be specified and built. Finally and importantly, aspects of the solution are harvested for subsequent reuse. The expected benefits from this approach include lower risk and cost, faster time to market, and an improved knowledge base for reuse in subsequent projects.

Choice and prioritization. Based on our analysis and the guidelines provided by our industry representatives, we selected the following five technical reference architectures:

- Thin-client transactional
- Collaboration
- Business intelligence
- Call center
- Mobile computing

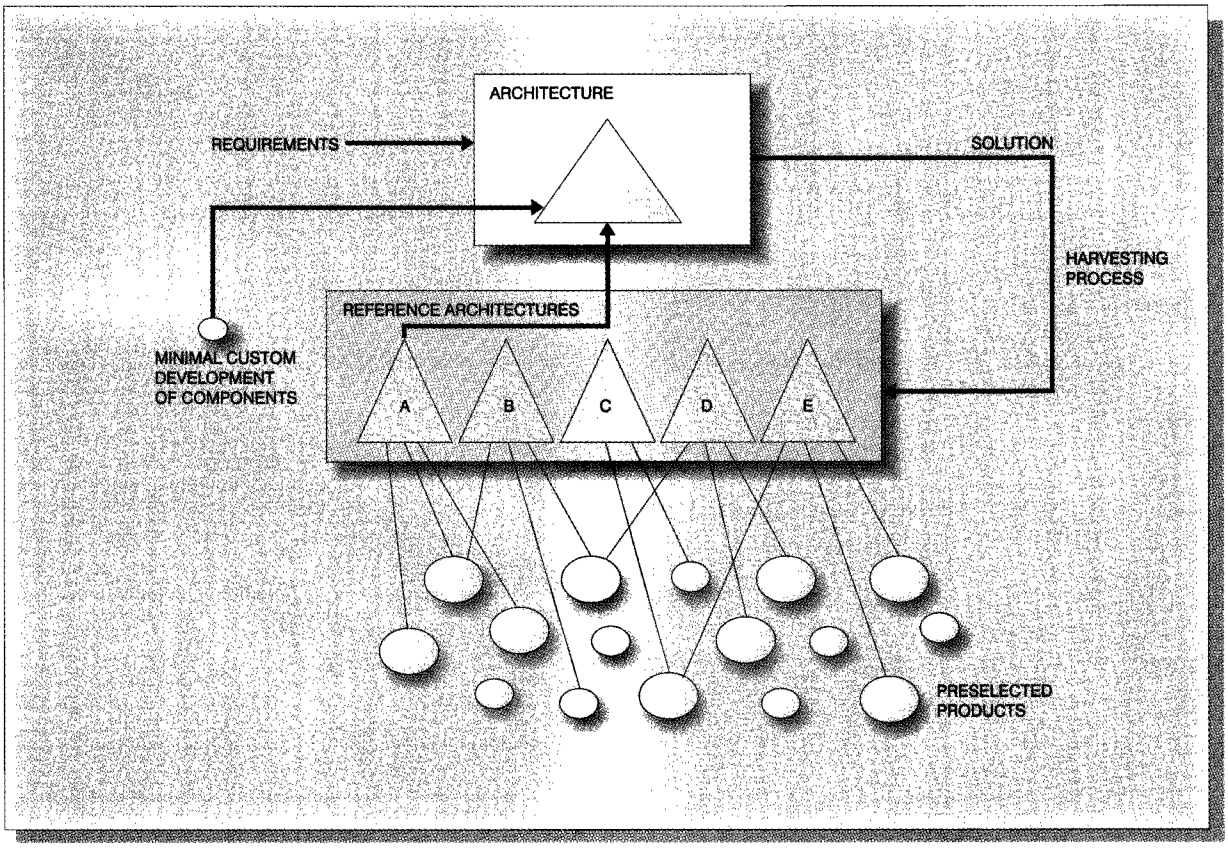
Thin-client transactional. This technical architecture addresses the need to do enterprise-scale administrative business, for example:

- Customer sales and service
- Order processing
- Claims processing, loan origination, etc.

It does not support, for example, solutions requiring real-time control of equipment. Its purpose is to support the business need of doing enterprise-scale commerce (as contrasted with business intelligence or collaboration) over the Web or via network-connected workstations. The essence of this category is the need to use highly secure, highly scalable transaction processing via this new channel.

Collaboration. The purpose of this architecture is to support the business need for collaboration. Collaboration occurs when two or more people participate in a task or interaction that relies on the use of common information. Asynchronous and synchronous collaboration services provide, to a dispersed set of participants, an effective means of working together and communicating with one another on topics of joint interest.

Figure 3 **The solution development process based on reference architectures provides a fast start. The feedback from the harvesting process is essential in keeping the assets fresh.**



Business intelligence. The name implies the exploitation of corporate data assets for competitive advantage. Business intelligence applications include executive information systems, decision support systems, and data mining applications. Operational systems collect transactional data in the course of day-to-day running of the business. Business intelligence users query and analyze data derived from these collected data and make decisions on how to improve business results based on these analyses. They may also make business decisions based on the discovery of previously unsuspected patterns of activity (data mining). Business intelligence applications are associated with "data marts" and "data warehouses."

Call center. The call center technical architecture supplements the transactional and collaboration architectures, and describes the overall pattern for implementing a "best-of-breed" call center that provides telephone access for customers to do business, accesses existing operational systems in the process of serving the client, and allows for the processing of transactions. It may provide fully automated access, e.g., via an interactive voice response (IVR) unit; it may provide a human interface in the form of a customer service representative (CSR), and it provides the infrastructure to merge the telephony and data processing worlds. An alternative name for the type of call center covered by this reference architecture is the "customer relationship management center."

Mobile computing. Mobile computing is a supplementary architecture to the transactional and collaboration architectures. *Mobility* is a term used by businesses to describe an environment that allows a worker to access information and perform his or her work "anywhere, anytime." A mobile user typically

utilizes various technologies, such as a laptop computer, a pager, and a cellular phone. Operation within the mobile environment may be connected to the network or disconnected from the network at least part of the time, and often most of the time. The mobile environment requires the development of applications that support the specific business needs of the mobile user.

Last, but by no means least, in order to avoid silos we defined an integrating architecture.

Enterprise technical reference architecture. This provides the "big picture" view where all subsidiary architectures are shown in an integrated manner. While it is unlikely that any of our enterprise customers will wish to implement such an all-embracing infrastructure, they will be reassured that the overarching architecture is consistent, so that new solutions can be added incrementally rather than producing more and more stovepipe systems.

The importance of legacy integration. Right from the start, it was clear that effective integration with legacy systems was critical for our work. Big as some of today's development projects are, 500 person-year systems are still small in relation to the huge investment that enterprise customers have made in their existing systems; for a large bank, these can be well in excess of 20 000 person years, quite apart from vendor-supplied operating systems or middleware. Furthermore, these legacy systems are typically mission critical, high volume, and in fact keep the enterprise running. There is no point in designing technical reference architectures for electronic commerce or business intelligence that do not take into account the importance of legacy systems.

In our reference architectures, we make use of a number of architecture styles that have been demonstrated to help with legacy integration. The styles are used in various combinations within the reference architectures. They include *wrapping,*[11] *hub and spoke*[3] (associated here with a *three-tier* pattern), and *back-end scripting.*[15] Wrapping in this context refers to the practice of hiding the existing legacy application interfaces behind an abstraction layer representing the programming model. Changes over time to the legacy system (for example, as it is replaced) can be made transparent to existing users. This wrapping will generally be done on a mid-tier server, with the legacy accesses "fanned out" to the third tier via appropriate adapters. To ease development of these adapters, which need to be specialized individually for each customer situation and need highly specialized knowledge of the back-end middleware, such as IMS* (Information Management System) or CICS, software providers such as Early, Cloud and Co.[16] with their MDp* (Message Driven processor) product have proved the effectiveness of back-end scripting as a boost to development productivity.

Quality standards. Our stated strategy is to harvest structures that have been successfully deployed at least three times in real engagements. This begs the question of what is meant by successful deployment, especially in those cases where no code has been delivered to demonstrate accurate functioning. For architecture engagements, we stretch a point and allow our criteria to be satisfied if the engagement has been completed to the satisfaction of both client and engagement manager, even if no executable code has been produced. This allows us to capture commonly recurring designs.

In our reference architectures we make use of architectural styles that help with legacy integration.

During the early stages, our team used subject-matter experts known to us to provide "certification" of the content of the assets that were abstracted and codified. To provide immediate benefits to our engagement teams, we have not always insisted on consistent notation. This is a pragmatic approach, but a more formal approach is required.

We have now defined explicit quality standards for each architectural object in our repository, and are in the process of setting up an architecture board. Board members are formally responsible for certain subject areas in the architecture—for example, security or workflow. Each architectural object will have a nominated "steward" from the board. At the same time, we are updating the content of the repository to conform to the standard semantics and notations defined by the IBM Architecture Description Standard.

Describing the technical reference architectures. A common standard is needed to describe these architectures. As stated by Shaw and Garlan, "It is now common practice to draw box-and-line diagrams that depict the architecture of a system, but no uniform meaning is yet associated with these diagrams."[8]

The need for a common standard. When we started to examine and then harvest technical reference architectures in 1996 and 1997, a fairly obvious but intractable stumbling block emerged—there was no commonly agreed upon standard for describing architectures. There certainly were effective architecture and infrastructure design methods in IBM (for example, the Enterprise Technical Architecture method, and the infrastructure design (ISD) method known by its acronym WSDDM [worldwide solution design and delivery method]-ISD), but the terminology was not common between these methods, and semantics and notations were not always precise. This affected us in two ways: first, it made existing architectures hard to understand, and second, we needed to describe our new technical reference architectures so that they could be reused by IBM practitioners.

Fortunately, at the same time that we were wrestling with this problem, our colleagues in the IBM Global Services North American SI/AD (Systems Integration/Application Development) practice began an initiative to converge several existing methods, including the object technology method (known by its acronym WSDDM-OT), WSDDM-ISD, and others. The new converged methods needed common semantics and, most important from our point of view, they were based on work products. In other words, their primary focus is on the artifacts to be produced during the development process (for example, operational architecture models), rather than the step-by-step processes to create the artifacts. From their point of view, a technical reference architecture is simply an instantiated, linked set of architecture work products that is appropriate for a particular purpose. From an asset-based point of view, this new converged method can be considered as a key part of the common formal methods layer in Figure 2.

In 1998 we started to work directly with the SI/AD initiative to help define a new common architecture description language. The first results of this work are described in an accompanying paper,[9] and have been used to provide the basic architectural expressions in this paper.

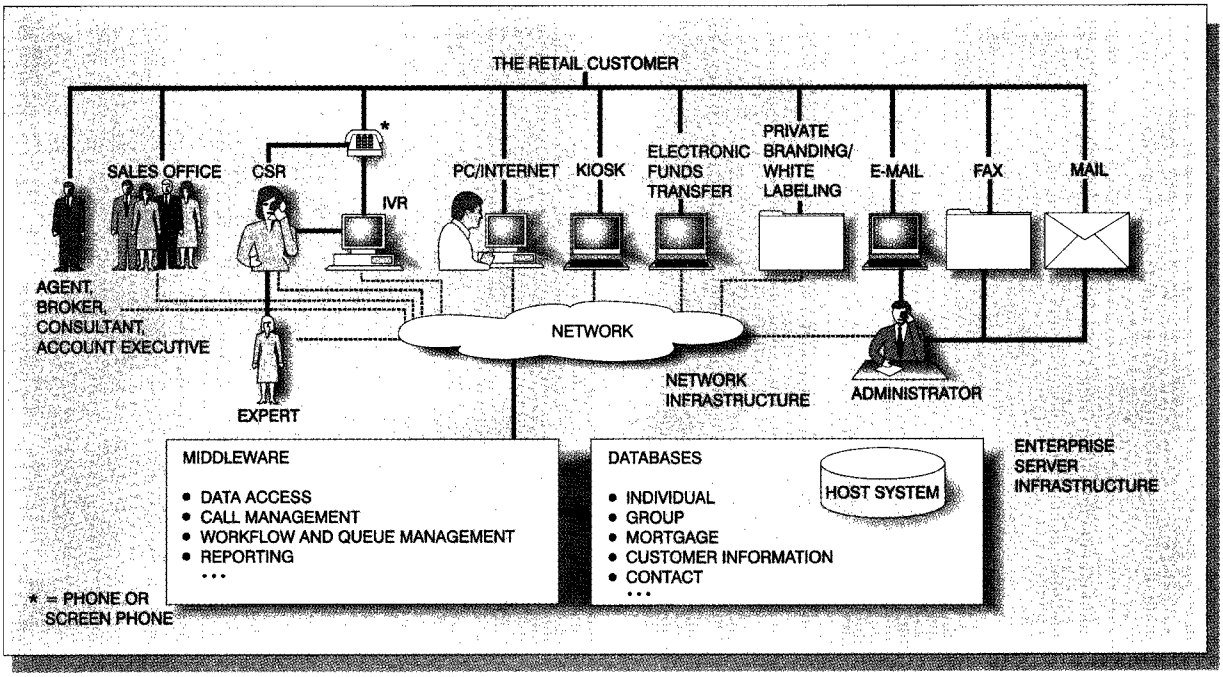
Additional requirements for the asset-based approach. Experience over the last few years, in IBM as elsewhere, has shown that merely filing assets in a library will not promote reuse. There is a real danger that such libraries will become "junkyards." Assets need to be planned, managed, and supported, as Jacobson et al. point out.[4] In the same way, architectures stored for reuse need to be given additional attributes that might not be needed for "from-scratch" development.

In our work to date, we have identified the following additional attributes as key to the asset-based approach: defined elaboration points, context definition, design guidance, and associated nonfunctional requirements.

We observed in our surveys that when asked to describe their architectures, project architects often showed us documents and diagrams with a very physical feel to them. Product names such as CICS, MQSeries*, Microsoft Windows**, and MDp were scattered throughout the diagrams. Presumably, at some earlier stage the logical architectures implemented with the physical products had existed, but they had somehow been lost. From the reuse point of view, although it is excellent news if an existing physical architecture (by which we mean one prepopulated with vendor middleware and operating system products, laid out over specific hardware nodes) can be directly reused, the chances of this happening are not high, because the environment varies so much from one enterprise to another.

So from a reuse perspective, it is vital that the technical reference architecture is described at several linked levels of abstraction. In general, the chances of reuse improve as the abstraction level increases, although at the same time the value of the asset in any individual engagement diminishes. We currently envisage three levels of abstraction, which we call *elaboration points*.[9] Among other advantages, this approach offers architecture teams several different potential entry points for reuse. At the time of writing we use the terms initial, logical, and physical to describe these different elaboration points, though this terminology is not completely satisfactory and is subject to change. We use *initial* to describe the architectural elaboration point with minimal constraints, *logical* for the set of designs that include network topology, including the differentiation of clients and servers, and *physical* for the stage where hardware architectures and operating systems are defined,

Figure 4 **We start with the contextual view to avoid the silo trap. Here we show retail customer access points for an insurance company.**



though not for example the number of hardware engines required.

Each technical reference architecture is linked to a *context*, which describes the purpose and requirements for which it is intended. Included with the context information are a number of informal *contextual views*, which represent different aspects of the existing and to-be system. These have been shown to be vital in communicating the essence of the proposed solution to the customer management and focusing on appropriate aspects of the architecture. Among other things, they are a key vehicle for helping to shape the requirements toward a standard approach, because they present examples and rationale from successful projects in similar customer environments. Contextual views currently cover the following aspects:

- Linkage between business model and IT architecture
- Topology views
- Access point views
- User desktop views
- Collaboration view
- Electronic mail view
- Mobile computing views
- Information system views

An example of an access point contextual view is shown in Figure 4.

It frequently happens that the customer's stated requirements could lead accidentally but inexorably to the development of yet another silo or stovepipe system. By looking at the to-be solution at high levels of abstraction, these dangers can be identified and the requirements modified.

These contextual views, which are a key part of technical reference architecture selection, are of particular value when vendor and prospective client get together at the proposal stage. Consider the following dialog, which is adapted from Bass et al.[5]

CUSTOMER (pointing to a thick stack of paper): "Here are my requirements. Can you meet them?"

IBM SALESPERSON (fondly remembering the good old days when all that was required was a smile,

a nod, and going back to the branch office with a fat contract in hand): "I'm sure we can. However, I should tell you that if you were to relax this requirement here, and this one here just a bit . . . have a look at this contextual view of our call center reference architecture" (pulls a set of charts from his briefcase). "You are asking to place your customer information file in the call center, but you can see from the chart that other access points that you mentioned to me really need to have that file located somewhere shareable . . . what most people seem to be doing these days is placing the file on the enterprise server. If you are flexible on these requirements maybe we would be able to satisfy them using a straightforward variant of our standard solution architecture."

CUSTOMER: "Good for you. How will it help me?"

IBM SALESPERSON: "To meet your requirements as stated will cost $20 million, take three years, and you will have a system unlike any other. There is additional risk because no one has done it like that before. On the other hand, to meet the slightly modified requirements will take 18 months, cost $10 million, and you will be using components that have been proved in practice many times before. It might even be a better solution for you. Which would you like? It's completely up to you."

CUSTOMER (who has never had a vendor offer a choice like this or been made to understand how much the "special" requirements actually cost, so that their worth can be judged): "Really? I will take the modified version. See you in 18 months."

Another learning point from IBM's experience in attempting asset-based reuse is that even experienced architects want to understand the reasons that lie behind the choice of particular aspects of the technical reference architecture, before they reuse it. There are strong reasons for this, including the sensitivity of the architecture to changes in the requirements, compared with requirements documented in the context for the architecture. Less experienced architects also need good advice when examining the viability of a technical reference architecture for reuse. For these reasons, we attach a *design guidance* attribute to all our architecture assets. Design guidance is provided in context and helps architects to assess the suitability of the particular artifact for use in a particular situation, and its sensitivity to changes in the requirements from those assumed for the reference artifact. Its purpose is to assist customization of the asset while maintaining its integrity.

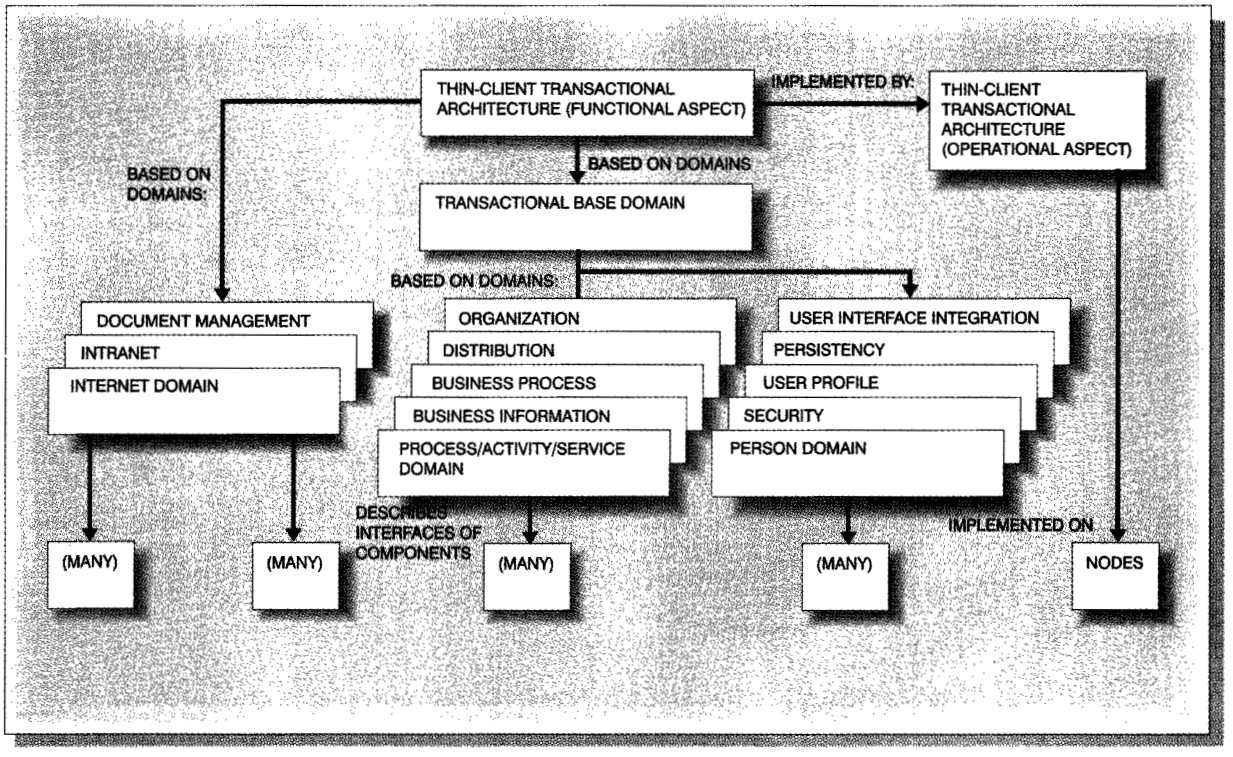
The *nonfunctional requirements*, and other qualities that are intended to be met by the technical reference architecture, need to be attached to the architecture constructs at various levels of granularity, ranging from the overall context to the specifics of particular software components and particular hardware nodes.

Additional requirements for large-scale enterprise solutions. Earlier, we noted the importance of avoiding silos in establishing the technical reference architectures. As the ESS technical architecture team began to codify and store fragments of the selected technical reference architectures, it was clear that to provide reuse, aid understandability, and avoid duplication, descriptions of the functional aspects of reference architectures needed to be based on descriptions of a number of smaller architectural fragments that we call *domains*. We define a domain as a subject area that defines a context for analysis and description of some aspect of an IT system. This is illustrated in Figure 5, which shows (an approximation of) the structure of the thin-client transactional technical reference architecture.

This reference architecture consists of a functional part and an operational part. The functional part is seen to be based on a "transactional base" domain, and other domains—Internet, intranet, and document management. The transactional base domain itself is based on a number of domains including security, persistency, and process/activity/service. This structuring of architectural assets allows other technical reference architectures to reuse architectural fragments—for example, the collaboration technical reference architecture also uses the Internet and intranet domains.

Large-scale enterprise solution development, as we have seen, poses serious challenges to development project managers. A good architecture, defined early, is a great help in many areas, including the separation of concerns and the structuring of development teams. Once the project size exceeds 100 person years, project organization becomes a critical success factor. Robert Prins[17] tackles these issues well, and in particular promotes a layered architectural style that we call process/activity/service and that matches well to our observations of actual architectures from several projects, particularly those dealing with enterprise workflow and call centers. In this

Figure 5 Structure chart for the thin-client transactional technical reference architecture

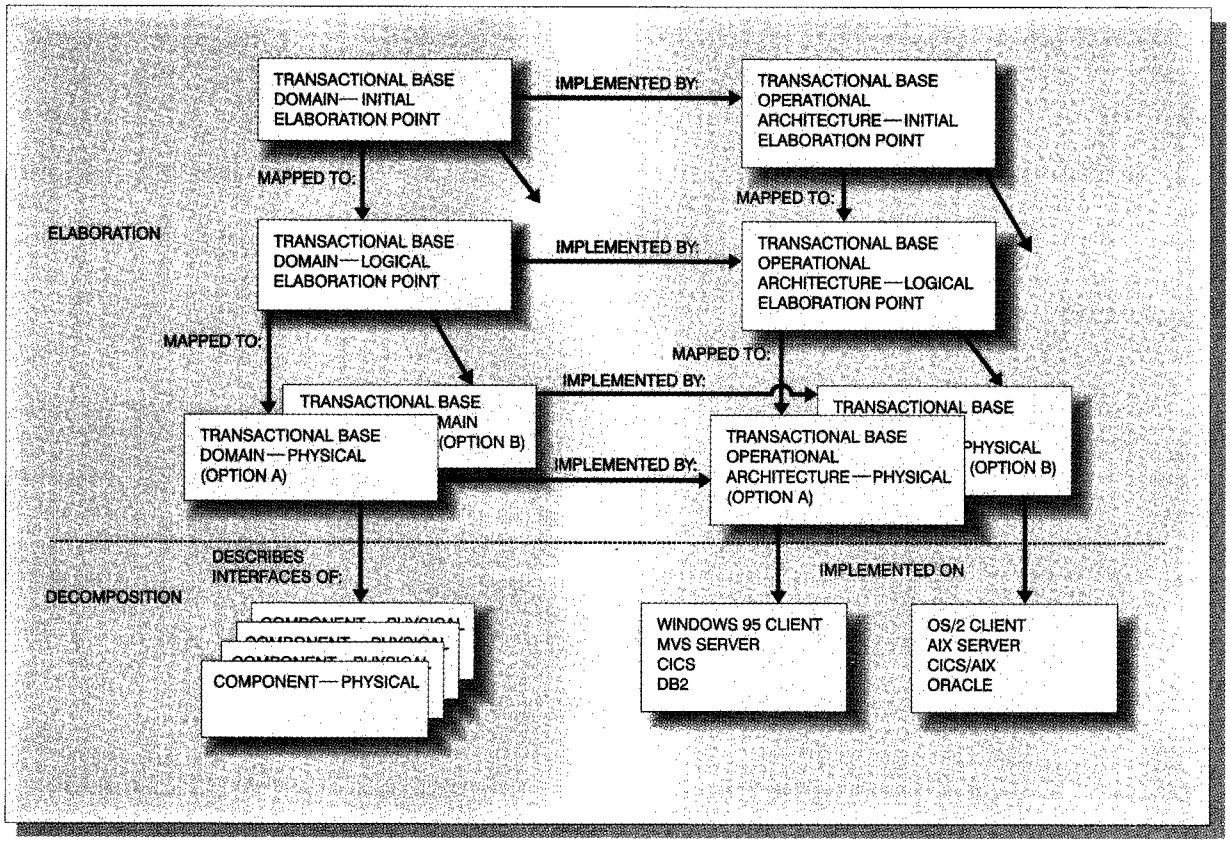


structure, the process elements are extracted from the relatively invariant underlying business functionality, and are themselves separated into enterprise-wide processes, with appropriate routing to roles, and desktop scripting, frequently seen in call center implementations under the terms "scripting" or "workflow." The project can be organized around these structures and work can proceed in parallel. This approach is well-suited to environments like insurance, and those where the process requirement is rapidly changing in a volatile business environment. It helps not only with initial development, but with the subsequent life of the system in its maintenance phase. It has been shown to fit well with application architecture development by other IBM departments providing insurance solutions for the enterprise marketplace.

Scope includes both functional and operational aspects. Personal experience, and observations from our project surveys, convinced the team that enterprise-scale technical reference architectures could not be regarded as adequate to support asset-based development unless they handled both functional architecture and operational architecture aspects. Why? We saw too many examples where elegant software architectures were "passed over the wall" late in the project to the "infrastructure architects" and proved impossible to implement in a way that met the non-functional and other quality requirements for the system. In other projects the infrastructure architects did a great job on infrastructure design, but there was insufficient rigor in definition to support a seamless integration with the developers building the application; expensive project time was then spent late in the project trying to patch the two parts together.

None of this is to say that there should not be separation of concerns in describing the architecture—quite the opposite, as we discussed earlier in describing the work of Prins. The point here is that the application development architects and the infrastructure architects need to speak the same language in terms of architectural descriptions and to understand the key integrating mechanisms (like component placement on nodes, and the shared interest in

Figure 6 Products are mapped from physical, not logical architecture

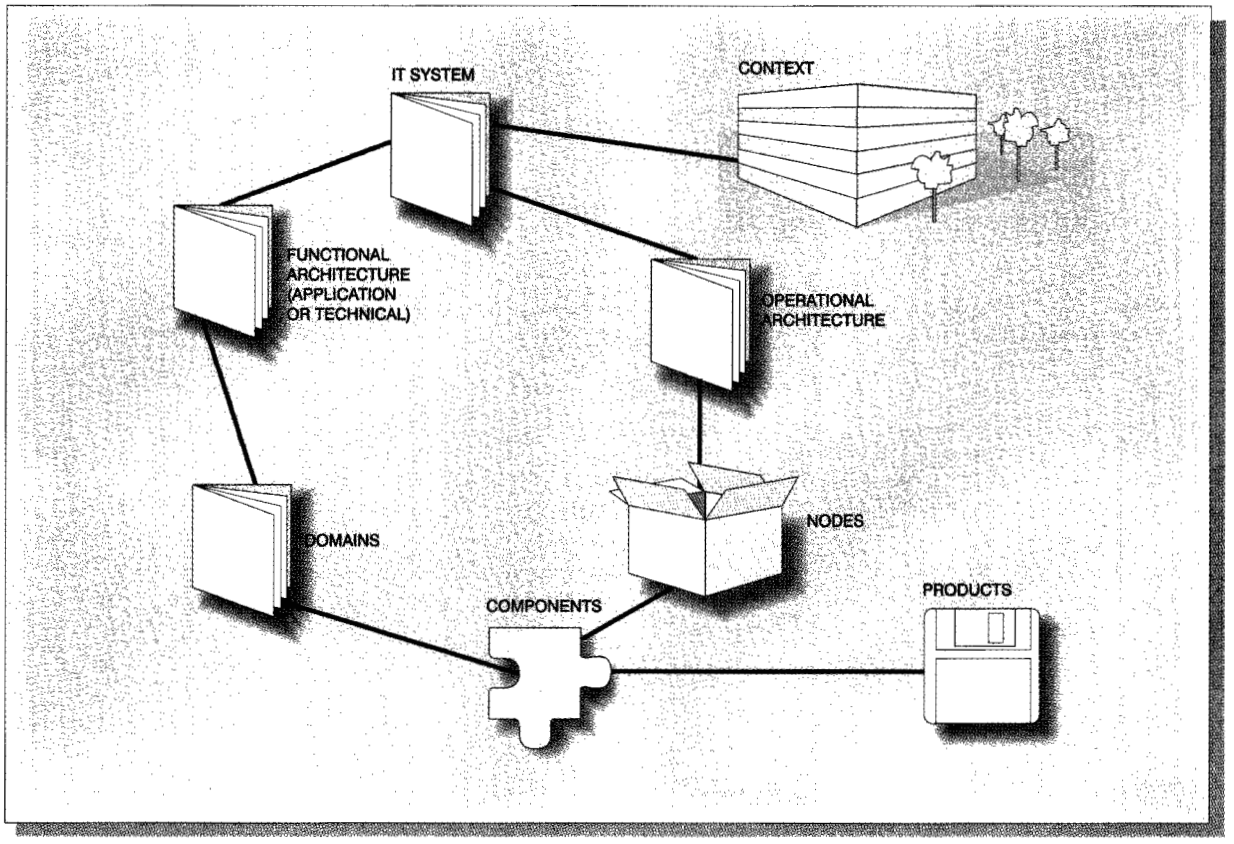


the semantics of control such as synchronization, locking, and transaction scoping). Once the common language is understood, each group is free to apply the special methods and tools in its own area of concern.

Product selection and logical architecture. The following words of R. Schulte of the Gartner Group are self-explanatory. "In many enterprises, 'architecture' means only a short list of standard products that have been approved for use . . . shortlist architectures do not offer application developers enough information to make successful design decisions."[3]

We saw earlier that from a reuse perspective, it is essential that the technical reference architecture is presented at several linked levels of abstraction. In ESS, product selection is included at the level of physical reference architecture, and there may be several physical reference architectures for each logical reference architecture, as shown in Figure 6. We recognize that different customers will have different existing physical IT environments and that some physical designs will fit better than others. However, we do not intend to instantiate many physical architectures per logical architecture, because for both customer and vendor, there is benefit in some constraint. From a vendor perspective, this helps product suppliers to focus on a small number of environments and hence improve quality and reduce costs, and helps service suppliers to focus the skill sets of their implementation teams on a preferred set of products, increasing staff flexibility and improving the ability to deploy while reducing costs. From a customer perspective, a gradual move to standardization has similar benefits and, to some extent, reflects the desire of many customers to set and follow a strategic direction.

Figure 7 The key metamodel artifacts used in describing ESS reference architectures



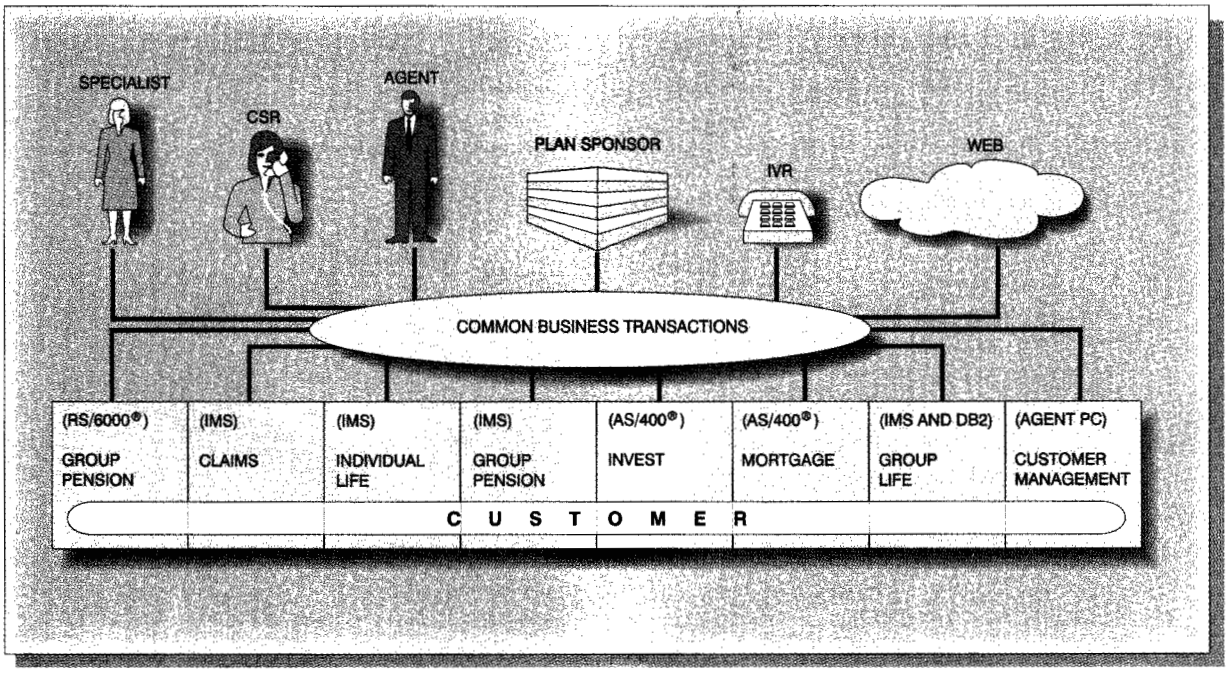
In Figure 6 (which illustrates concepts rather than describing actual ESS content), it can be seen that the transactional base architecture for this example offers two options at the physical elaboration point for a particular logical architecture. For client operating system, in the first option we see Windows 95** and in the second option OS/2* (Operating System/2*). Similarly, for server operating system we see MVS and AIX* (Advanced Interactive Executor); for transaction monitor, CICS and CICS/AIX; and for database system, DB2* (DATABASE 2*) and Oracle**.

A requirement placed on us by our industry representatives was that ESS should not confine itself to IBM products. Where other vendor products matched the architecture, and were widely installed, they should be considered for inclusion in the technical reference architectures.

Architecture description language. We can now summarize the main artifacts used in describing technical reference architectures. (Please refer to the accompanying paper[9] for further insight.) Figure 7 provides a simplified diagram of the metamodel. This diagram shows a simplified view of the relationships among many of the artifacts we have described, for example the placement of software components on nodes and the mapping between software components and products. As we have seen, the actual metamodel is more complex, for example in restricting product mappings to particular elaboration points in the architecture.

Tool considerations. As we started to collect and codify technical reference architectures in 1996, we initially used documents and presentations to describe them. It became clear almost immediately that this was not the right approach for building a reusable repository of architectural assets. With more than 200 components and 60 nodes, version control via documentation was almost impossible, and standard

Figure 8 **Information system view of business function. In a three-tier client/server model, the common business transactions are placed so that they can be used from all access points.**



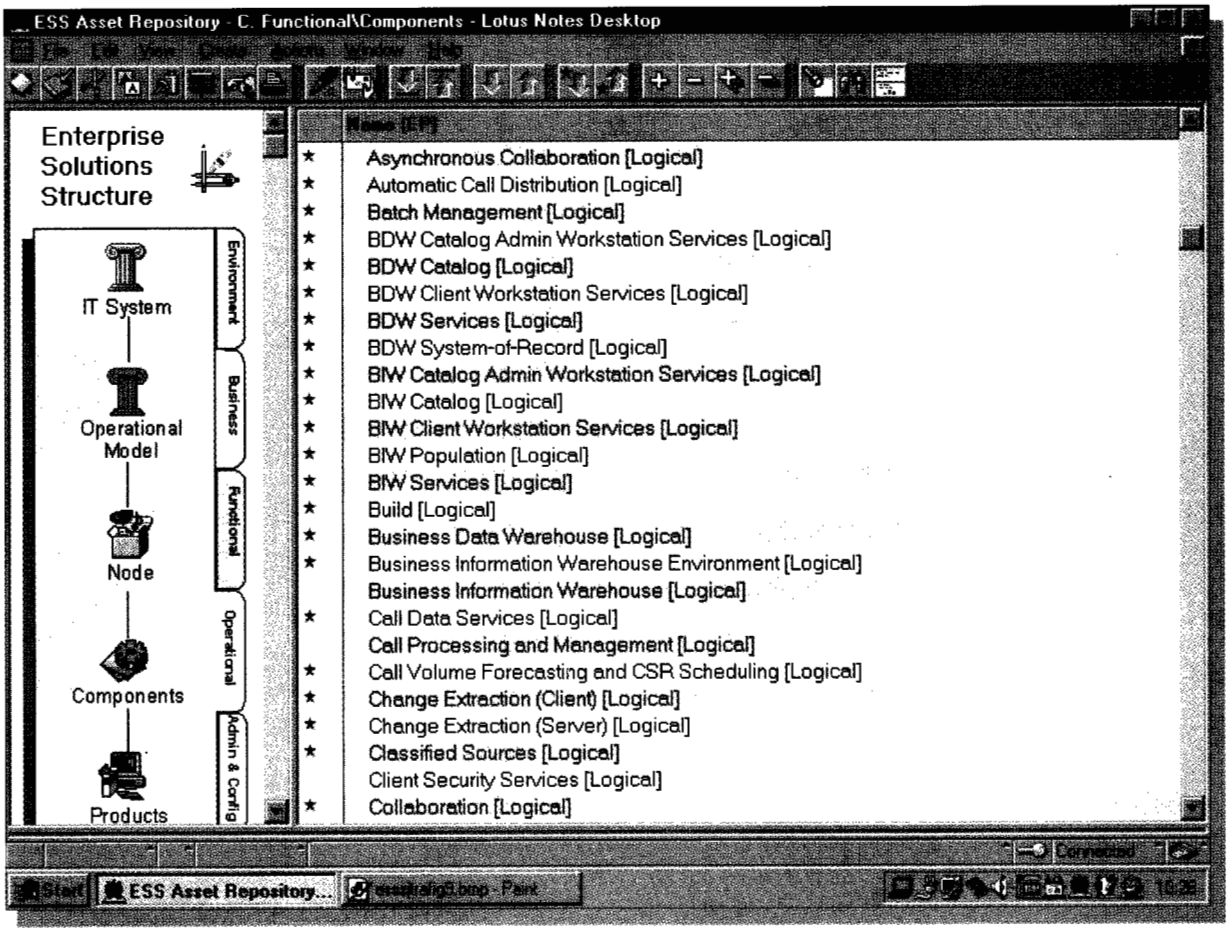
documents did not encourage rigor in semantics and notation. Gaps in the content were difficult to spot.

We made a decision in 1997 to document formal component models in Rational Software Corporation's Rational ROSE** modeling tool, using notations based on UML (Unified Modeling Language)[18] where this was appropriate for the component described. In addition, we developed a Lotus Notes** database to support the rest of our artifacts and to provide an indexing mechanism into the ROSE models. Notes facilities allow us to structure material in a relatively formal way to support our semantics. Notes has excellent replication capabilities for distribution of the material to professionals (in IBM, the Services and Global Industries professionals have standardized on the use of Notes clients and IBM laptop PCs). It also allows us to store existing documents and presentations for use with clients. Where appropriate, we "cut and pasted" ROSE diagrams into Notes artifacts for easy access by our professional teams.

The combination of Lotus Notes and Rational ROSE tools has proved effective. We have a few problems, which will be addressed as we gain more experience. These include the requirement to be able to use selected, up-to-date architectural artifacts in client workshops from time to time; IBM presenters typically use transparencies for this purpose, and this makes it difficult for us to manage copies. However, we are reluctant to begin potentially complex and expensive tool development until we gain more experience in architecture reuse.

During early architectural engagements using ESS assets, in a number of cases the clients have concluded that a structured Notes database is a good way for them to store and maintain their own set of tailored architectural descriptions, rather than base their architecture definitions on standard word-processed documents. This is particularly helpful when the IT department is distributed over several locations, where Notes replication facilities support local update but provide at the same time a consolidated view. It allows the architectural descriptions to be kept fresh while particular domain descriptions are delegated to relevant departments. Further information about these experiences is provided in another paper in this issue.[19]

Figure 9 Sample list of logical components in the ESS technical reference architectures

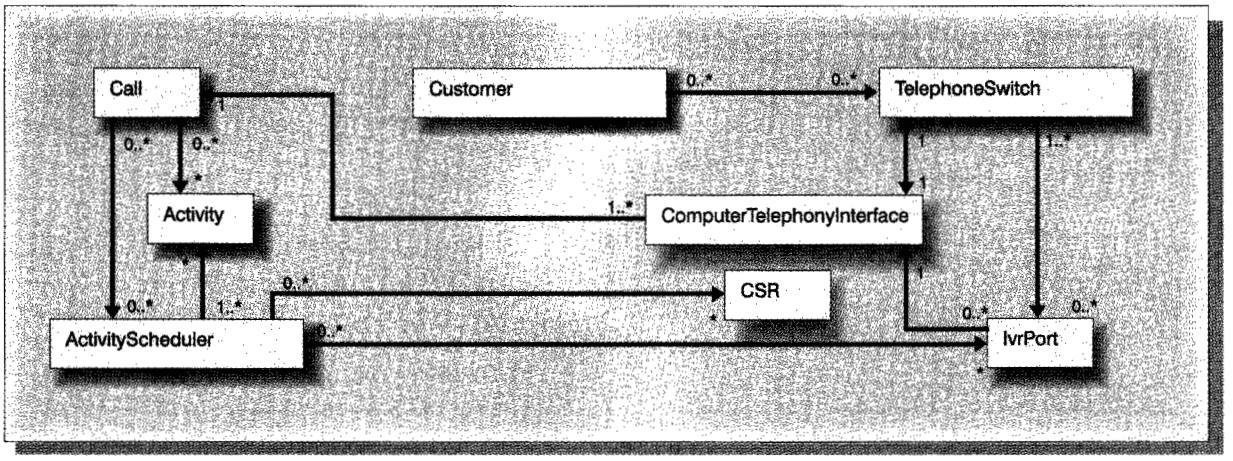


Some examples of content. In this section we describe some examples of architecture artifacts that are part of the ESS technical reference architecture content as we write this paper. By the time it is published, there is a high probability that the content, and possibly the notations will have changed. These examples should not be viewed as definitive; we offer them to aid understanding and provide additional insight. Some details have been omitted.

It is not our intention to explain how the assets are used in any particular case, and it is outside the scope of this paper to describe how solution development methods make use of the architectural assets. However, the assets are structured to aid the architect in his or her work—examples include the design guidance attribute attached to each relevant artifact, and the use of contextual views to aid communications (a key part of the architecture business cycle described by Bass et al.[5]).

Contextual views. In Figure 4, we showed an example of a contextual view that focused on the access points for the required solution. Experience has shown that early views of the to-be (target) IT system are of great value in helping clients to understand some of the important business implications of architectural decisions. For example, we have found that a chart similar to Figure 8 helps clients to appreciate the benefits of the "hub and spoke" pattern to support an integrated view of their customer (a holistic view, rather than separate product-

Figure 10 (Part 1 of 2) Component relationship diagram from the telephony domain



holder records) through use of a three-tier architecture.

Components. Figure 9 lists some of the software components defined in our technical reference architectures. The list is taken from the ESS Lotus Notes asset repository and shows components at the logical elaboration point—most of those shown are used in the call center or business intelligence reference architectures.

When harvesting and generalizing components, we need to be careful about the level of granularity to which we decompose the functionality. A component is not the same thing as a class in object-oriented terminology. In many cases, there are existing vendor products that could be used to implement each of these components, and this serves as one guide to the lowest level of component description that we provide. However, some vendor products are really suites of products that need to be located on multiple nodes. In this case the internal structure can be important in creating the operational or functional aspect of architecture; if so, we break down the component models to a lower level of granularity. In Figure 9, the component "BDW System-of-Record" encapsulates a lower-level component "System-of-Record Access" (not shown in the figure), and it is this lower-level component that is linked to a vendor product at the physical elaboration point of the business intelligence technical reference architecture.

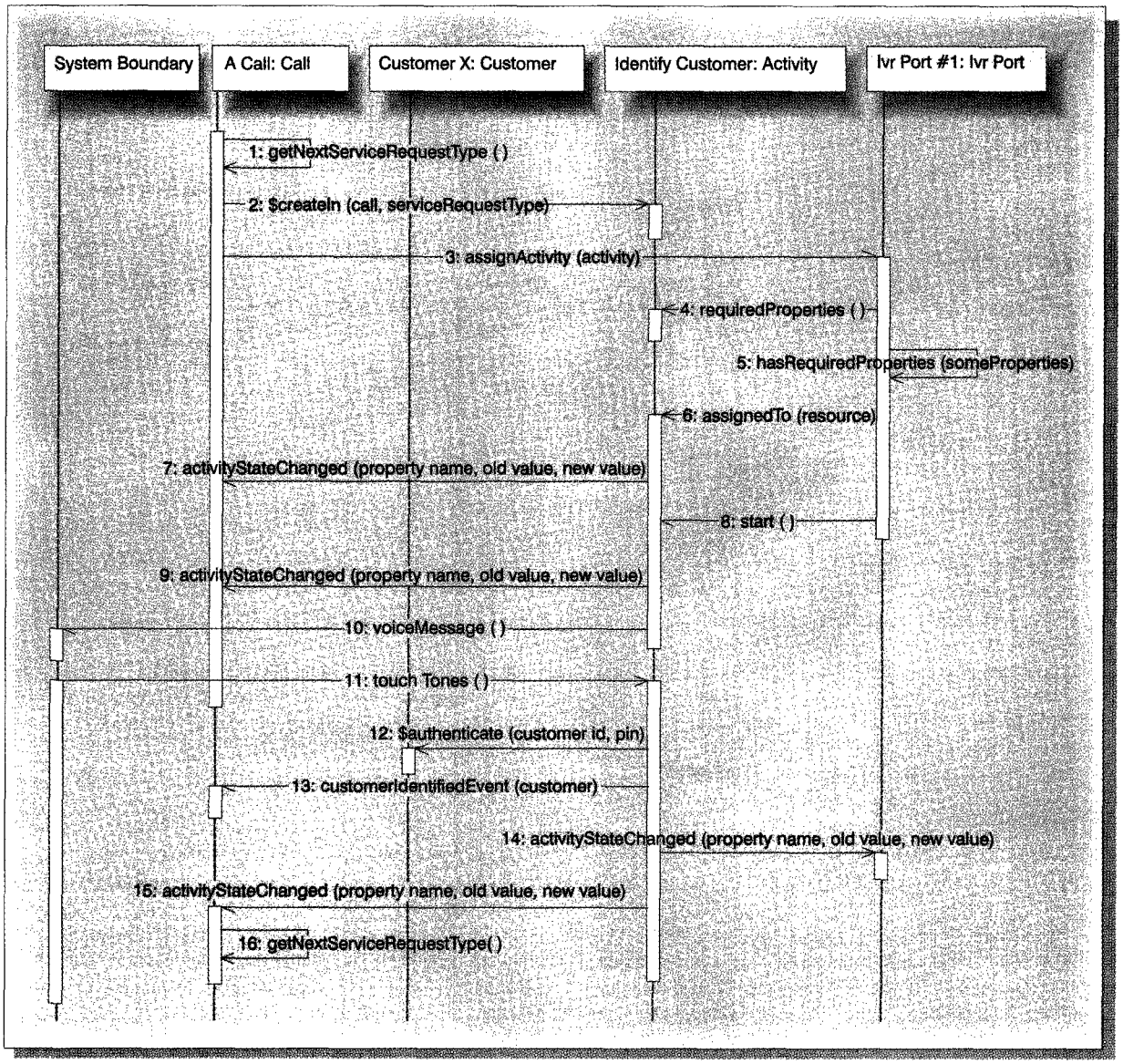
Component models. Figure 10 is a simplified illustration of the component model (showing both a component relationship model fragment and a component interaction model fragment) from the telephony domain, which is part of the call center technical reference architecture.

In the component relationship diagram, we are showing the main usage relationships, together with their cardinalities. For example, the Customer component makes use of the TelephoneSwitch component. Please note that the ComputerTelephonyInterface component, despite its name, is a component in its own right that provides the functionality to interface between a TelephoneSwitch component and a computer telephony integration application. Its responsibilities include providing an event channel from a switch to an application, providing a command channel from the application to the switch, and providing the application with an interface that is independent of the type of switch.

In the component interaction diagram, we see various messages being exchanged between the identified component instances during the collaboration *Identify Caller*, using an interactive voice response component. A *collaboration* is defined as an occurrence of a sequence of operations that realizes a use case scenario.[9] The conventions followed in the diagram are those of UML; for example, a half arrowhead denotes an asynchronous message.[18]

As previously mentioned, from the surveys of real-life engagements in 1996 and 1997, and from other sources, for example the work of Robert Prins,[17] we

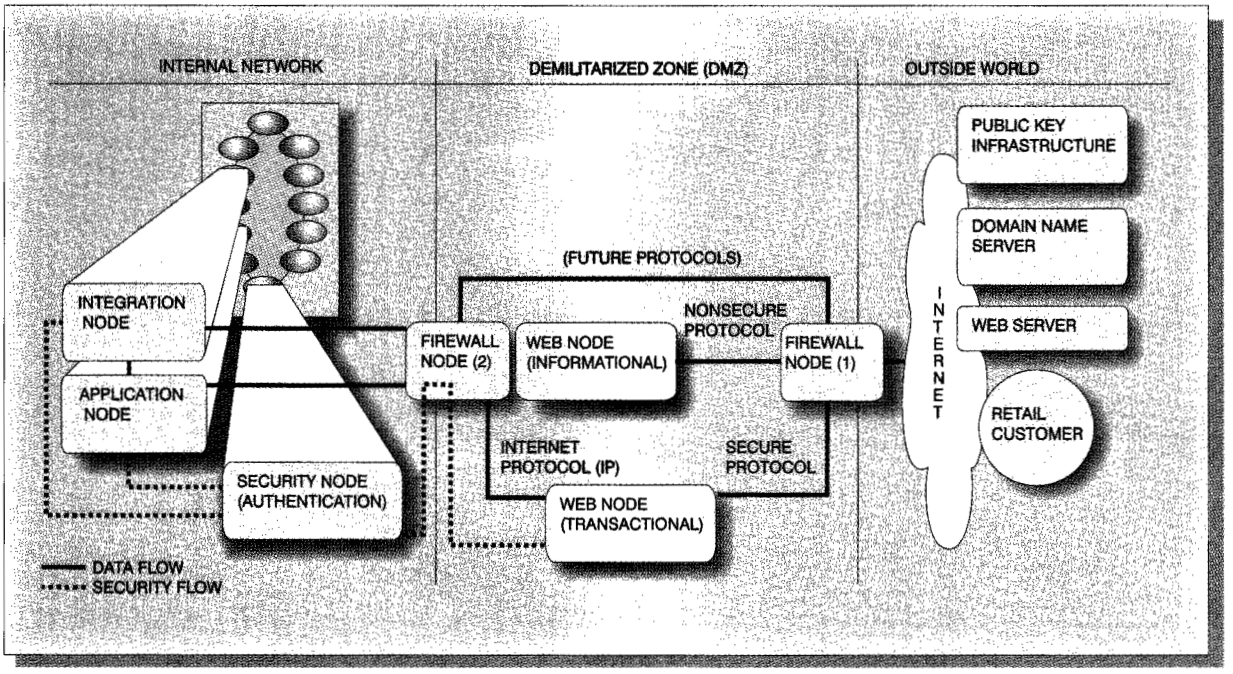
Figure 10 (Part 2 of 2) Component interaction diagram for collaboration Identify Caller



identified an architectural pattern that we call process/activity/service. This links the world of intracompany administrative workflow, through desktop scripting, to specific atomic units of processing that are relatively invariant over time in a particular IT system. This is particularly relevant in call center applications, and the telephony domain is closely linked with the process/activity/service domain.

Logical operational architecture. Figure 11 shows a simplified fragment of the operational aspect of the thin-client transactional technical reference architecture, which focuses on the security domain within that architecture. Associated with this diagram is a *walkthrough* (not illustrated), which has similar notation to a component interaction diagram but identifies nodes, so that the service level characteristics

Figure 11 **Simplified fragment of the logical operational architecture diagram from the thin-client transactional architecture**



and other qualities of this architecture, such as security, can be discussed and analyzed.

Nodes. Figure 12, taken from the ESS Notes database, lists some of the logical nodes in the ESS technical reference architectures and provides some idea of scope and content for the integrating enterprise technical reference architecture.

Figure 13 shows some of the logical nodes selected for the thin-client transactional technical reference architecture, as a subset of the nodes of the enterprise architecture—selected nodes are highlighted. Some of these nodes (for example domain name system) will be reused in other reference architectures, while others (such as integration) are unique to the transactional technical reference architecture. Nodes that are not highlighted include a group devoted to the call center reference architecture (for example CTI [computer telephony integration], PBX/ACD [private branch exchange/automatic call distribution], and IVR), and others associated primarily with the collaboration architecture.

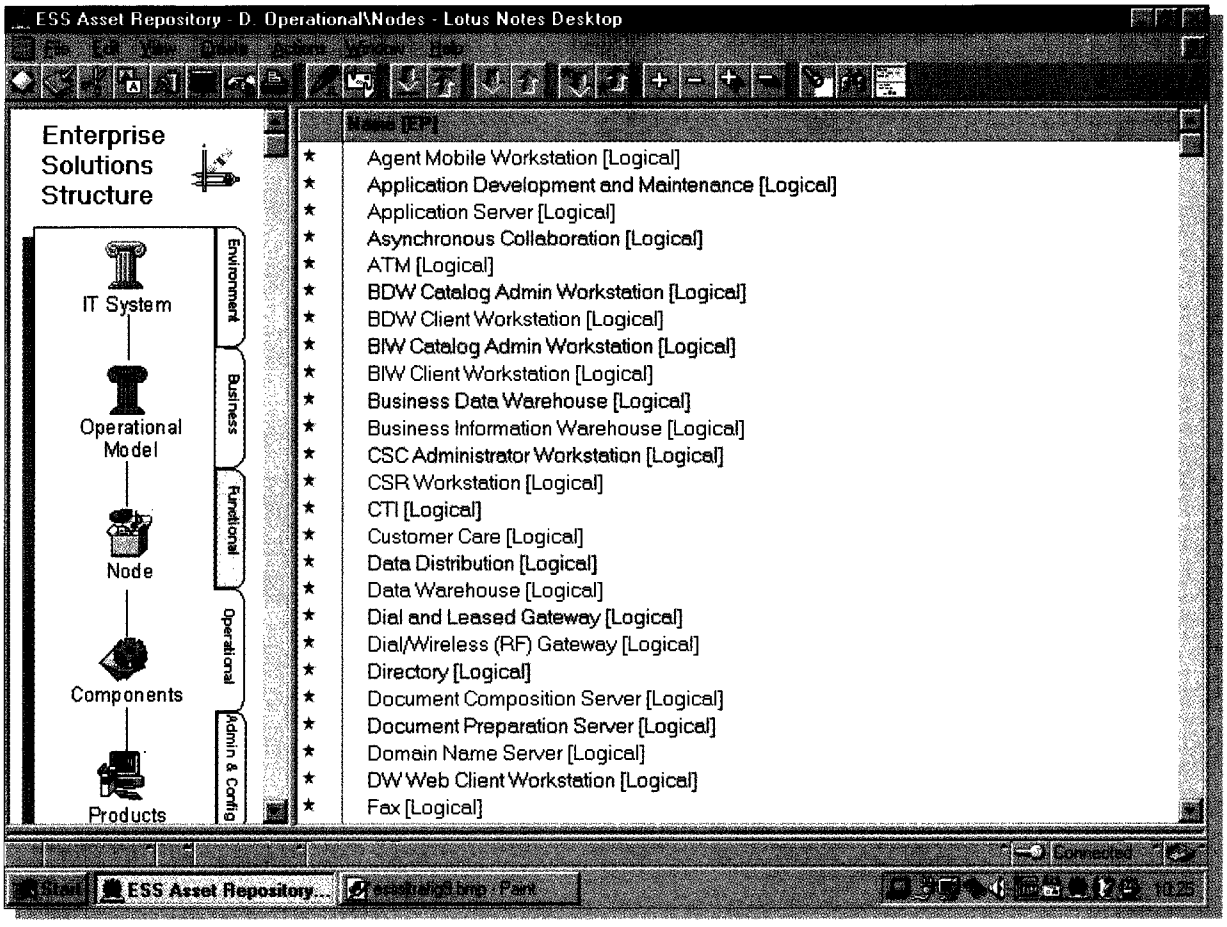
One of the nodes in the enterprise architecture is "integration," or integration server. This relatively complex node has various components placed on it, as shown in the simplified view in Figure 14.

The integration server is an extremely important part of the thin-client transactional technical reference architecture, as it represents a key logical tier in a three-tier architecture. As we have harvested more instances from engagements, we have extended our descriptions of the integration server and the components that are located on it, and we continue to develop our understanding of this area.

Using technical reference architectures. Although the strategic aim of the ESS program is to improve effectiveness in building IT solutions, we have been successful in using technical reference architectures to create IT architectures for clients. For this purpose we have used charts similar to Figure 15 to describe the reference architectures. This chart shows some of the key artifacts within the various reference architectures, and the arrows give some indication of the sequence in which the various artifacts are used during a typical engagement.

Our engagement teams used contextual views both to confirm understanding and to encourage a con-

Figure 12 Sample list of logical nodes in the ESS technical reference architectures



vergence to one or another of our technical reference architectures, if appropriate. Logical architecture models (primarily node descriptions and operational architecture models) were customized, and if necessary extended, as needed for the engagement, using the in-context design guidance provided as part of the asset set. From a consideration of the existing and to-be IT environments, corresponding physical architecture models were customized and extended, and product selection completed.

In some cases, clients were sufficiently impressed with the power of the semantic model that the engagement contract was adjusted and the required architecture was delivered with a supporting tool (either Web-based or Notes-based).

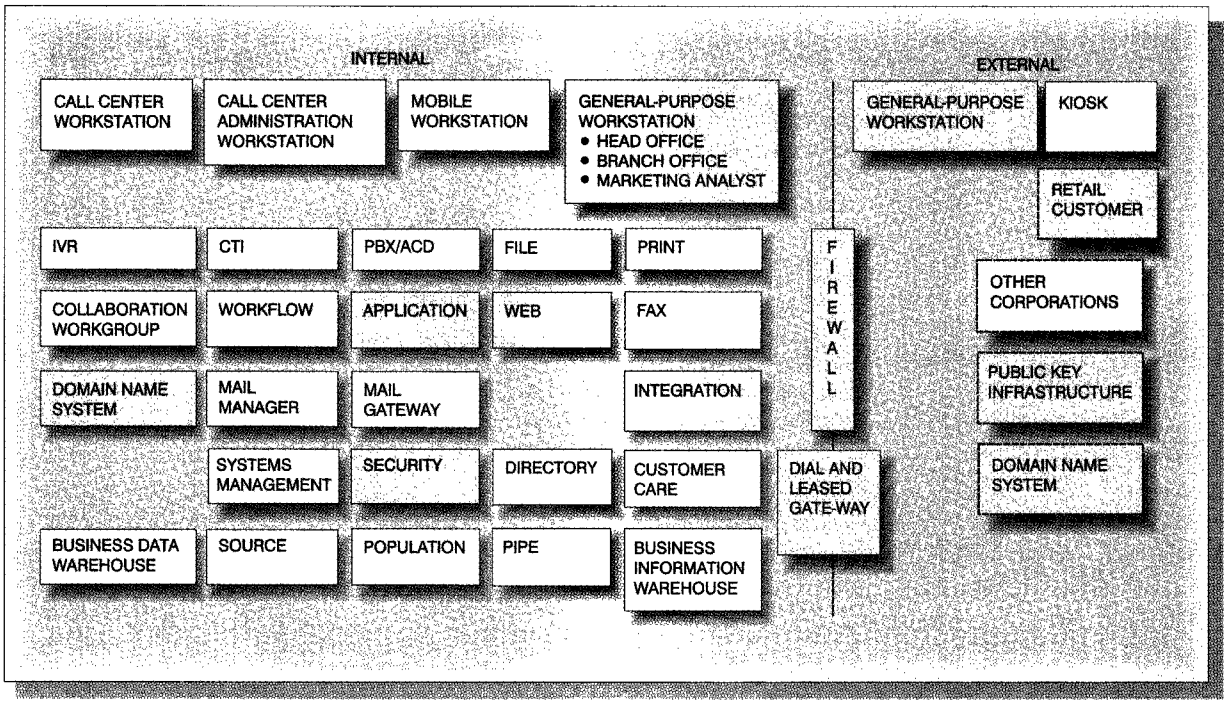
Throughout the engagement, the IBM team tracked the modifications made to the technical reference architectures, and where these changes were assessed as being of general value rather than specific to only one client, they were communicated back to the ESS technical architecture team for consideration as reference architecture enhancements.

For further information on our experiences in using the technical reference architectures on architecture engagements, see the accompanying paper.[19]

Concluding remarks

Existing methods of large-scale solution development and deployment rely too heavily on the "he-

Figure 13 Part of the node inventory for the enterprise architecture. Nodes in the thin-client transactional reference architecture are highlighted in green.



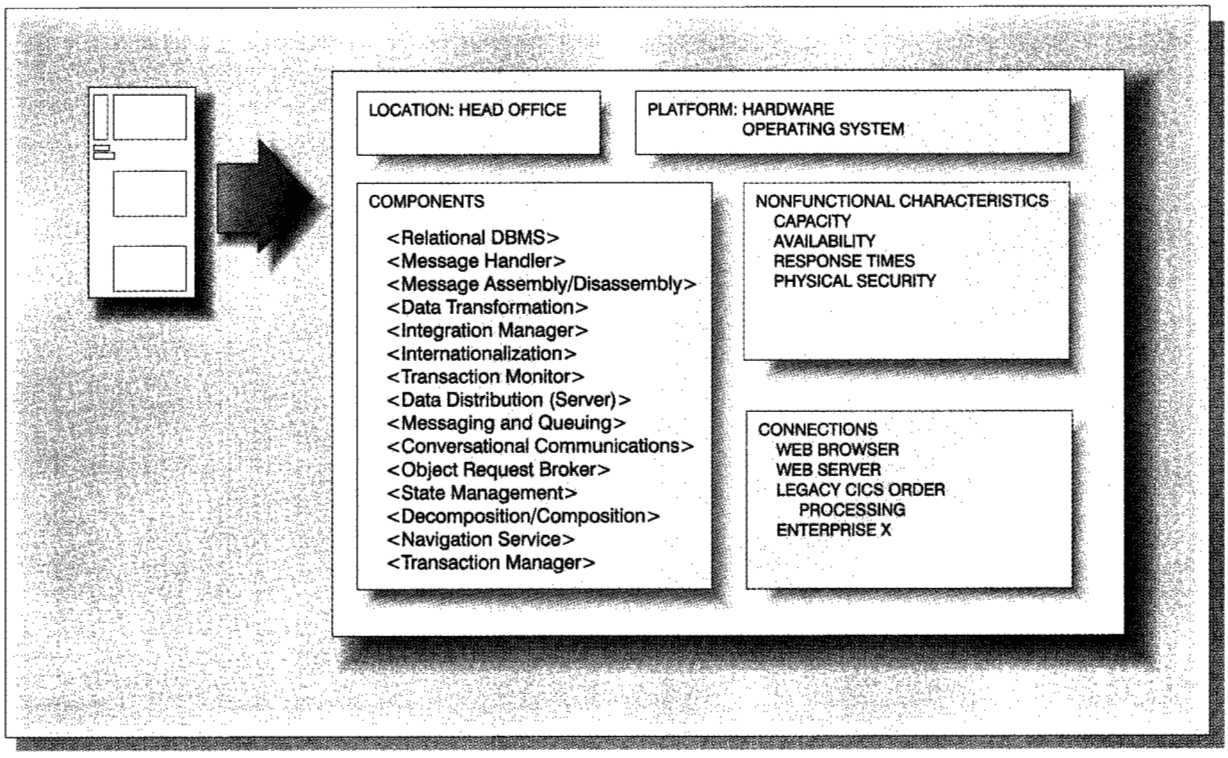
roic" approach and have proved unreliable. It is urgent that the IT industry move away from this handcrafting approach to development, and accept greater discipline and constraint in order to meet the needs of the businesses it serves. The concept of technical reference architectures, together with their supporting processes, provides a means for IT practitioners to build robust, scalable enterprise solutions quickly, with improved quality and reduced risk. In this paper we have shown how IBM has structured and built a constrained set of such reference architectures, harvested from successful enterprise consulting engagements; and we have indicated that our early experiences in deploying them in architecture engagements have demonstrated the approach to be effective.

The reference architecture approach presents its own challenges. Chief among them are first, the reluctance of the IT professional community to systematically reuse the work of others; and second, the corporate business model and organization changes needed to reward and support reuse and standardization. In today's fast-moving IT environment, the feedback processes supporting the reference architecture approach are crucial in ensuring the vitality of the architectural assets. With them, and with the wholehearted support of the practitioner community, the discipline of the market will ensure that the technical reference architectures are kept fresh and represent the best current practice.

One of the benefits in using technical reference architectures for solution development is to reduce risk, and we have mentioned some of the mechanisms, including in-context design guidance, that we use to help our practitioners customize the assets sensibly. But a good set of architectural assets does not avoid the need for architects to be properly trained, and successful implementation of the asset-based approach should include training in good work-product-based development methods, such as IBM's SI/AD method for its Global Services community, and appropriate subject matter expertise.

One of the themes underpinning our implementation of the reference architecture concept has been the need to integrate the operational aspect of ar-

Figure 14 Simplified description of the integration server node



chitecture together with the more familiar functional aspect. We have stressed that to be successful, a solution must be grounded in reality. Actual capabilities and limitations of hardware, operating systems, and middleware need to be considered right from the start. IT vendors must take and support this holistic view for the full potential of technical reference architectures to be realized.

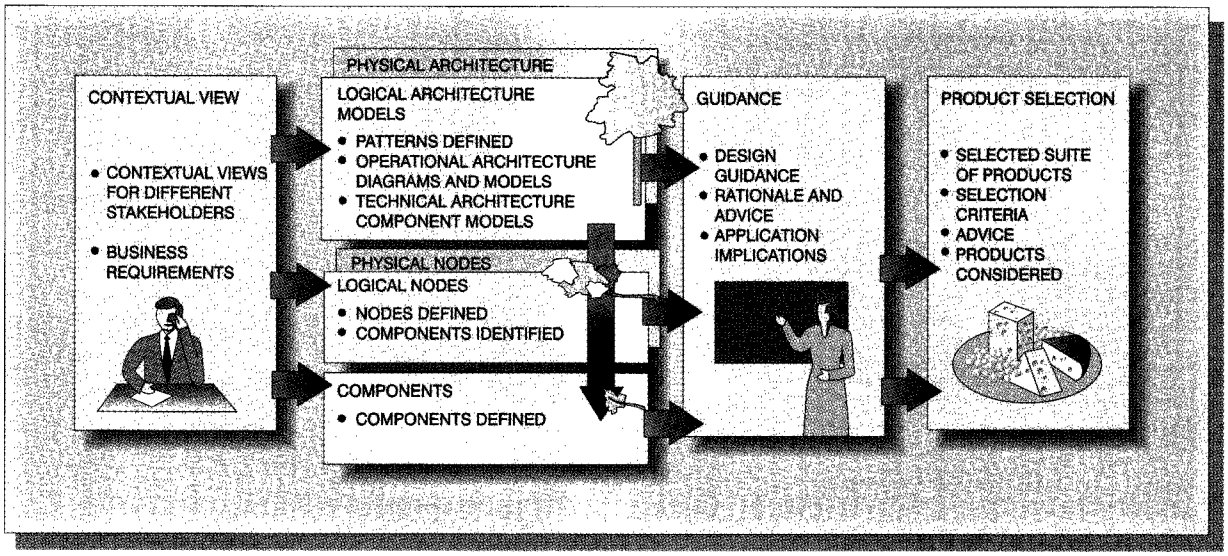
Acknowledgments

The codification of technical reference architectures is a complex task, and many professional colleagues have contributed to our work. We would like to recognize in particular Martin Cooke, Bruce Crossman, Jan Lock, John Rothwell, Philippe Spaas, and Robert Youngs, who have been part of our core team almost from the start. We also recognize our team colleagues Tim Barrett, Ian Charters, Steve Clarke, Steve Cook, Ralph Hodgson, George Hutfilz, Deborah Leishman, Doug McDavid, Silvia Pighin, David Redmond-Pyle, and Anders Stange.

Along the way we received invaluable help and encouragement from executives, "subject-matter experts," and engagement teams from many parts of IBM, including Jonathan Adams, Rick Ahlgren, Jim Amsden, Rock Angier, John Baker, John Black, Peter Bohnhoff, Anne Bomford, Marty Buskirk, Ron Buszko, John Cameron, Jean-Pol Castus, Alan Chivers, Chris Codella, Jim Davey, Ciaran Dellafera, Vince Devine, Alan Dreibelbis, Derek Duerden, Dave Ehnebuske, Don Ferguson, Patrick Fournery, David Gamey, Qing Ge, Bob Gray, Tina Harris, Philip Hausler, Ed Hood, Ton Ikink, Arjen Jansen, Keith Jones, Ed Kahan, Genie King, Nina Liang, Mike Lloyd, Leo Marland, Jim McGugan, Ross McKenrick, Julian Paas, Dave Parkhill, Maurice Perks, Emily Plachy, Colin Rous, Oliver Rye, Geoff Sharman, Dave Spencer, Howard Taylor, Phil Teale, Emeline Tjan, Thomas Wappler, Ray Wells, Neale Whyatt, and Roger Wright.

We would like to thank our colleagues Burnie Blakeley, John Fetvedt, and Robert Youngs for their help in reviewing this document. Finally, we would like

Figure 15 **Technical reference architecture assets cover a wide range, from patterns of requirements to product selection.**



to thank all those project IT architects "out in the trenches" who kindly gave their time to help us with our surveys. We hope that one result of our work will be that before too long, their life will be less stressful, less frustrating, and more rewarding.

Cited references and notes

1. A "silo" is a part of an IT solution that is not integrated with other parts. The name comes from the visualization of such a solution—the different applications stand apart from each other, like farm silos in silhouette against a prairie sky. Another name for this kind of solution is "stovepipe." A good explanation of stovepipes as an "antipattern" can be found in W. J. Brown, R. C. Molveau, H. W. McCormick, and T. J. Mowbray, *Antipatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, Inc., New York (1998).
2. In a fat-client solution, client systems perform most of the data processing operations. The data may be stored locally or on a server.
3. R. Schulte, *Architecture and Planning for Modern Application Styles*, Gartner Group Strategic Analysis Report SSA: R-ARCH-104 (April 28, 1997).
4. I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley Publishing Co., Reading, MA (1997).
5. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley Publishing Co., Reading, MA (1998).
6. E. C. Plachy and P. A. Hausler, "Enterprise Solutions Structure," *IBM Systems Journal* **38**, No. 1, 4–11 (1999, this issue).
7. B. Boehm, "Engineering Context," *First International Workshop on Architecture for Software Systems*, April 1995, Seattle, WA.
8. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, Upper Saddle River, NJ (1996).
9. R. Youngs, D. Redmond-Pyle, P. Spaas, and E. Kahan, "A Standard for Architecture Description," *IBM Systems Journal* **38**, No. 1, 32–50 (1999, this issue). This paper contains a glossary with more detailed definitions of terms used here.
10. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc., New York (1996).
11. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., Reading, MA (1995).
12. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, New York (1977).
13. K. A. Bohrer, "Architecture of the San Francisco Frameworks," *IBM Systems Journal* **37**, No. 2, 156–169 (1998).
14. Information about IBM Component Broker Series is available at http://www.software.ibm.com/ad/cb.
15. With "back-end scripting" the mid-tier server (the "hub") takes a request for information and decomposes it into a script of multiple, discrete tasks, each appropriate for an individual legacy application, and manages all of the associated workflow.
16. Early, Cloud & Company is now part of the new Customer Relationship Solutions (CRS) software company, an independent subsidiary of IBM. For more information see http://www.ibm.com/services/crs/index.html.

17. R. Prins, *Developing Business Objects: A Framework-Driven Approach*, McGraw-Hill Publishing Company, London (1996).
18. See http://www.omg.org for link to UML specifications.
19. T. Harris, J. W. Rothwell, and P. T. L. Lloyd, "Experiences in Reusing Technical Reference Architectures," *IBM Systems Journal* **38**, No. 1, 98–117 (1999, this issue).

Accepted for publication October 9, 1998.

P. T. L. (Tim) Lloyd *IBM United Kingdom Ltd., P.O. Box 31, Warwick CV34 5JL, England (electronic mail: tim_lloyd@uk.ibm.com).* Mr. Lloyd is a consulting IT architect in the IBM United Kingdom Object Technology Practice. He led the ESS Technical Architecture team from its inception in 1996, and subsequently became team leader for overall ESS development. Currently, he is lead architect for the ESS Operations team. He previously worked as enterprise systems engineer and solutions project consultant for several large IBM enterprise clients in the government, retail, and utilities industries, as technical architect on a large insurance solutions development project, and as regional specialist in various subject areas, including large systems hardware and software, storage systems, and office systems. Other areas of experience include the application of image processing and workflow in enterprise-scale solutions. He received a B.Sc. degree in theoretical physics from Manchester University.

George M. Galambos *IBM Canada, 1250 Boulevard Rene Levesque, Montreal, Canada H3B 4W2 (electronic mail: galambos@ca.ibm.com).* Dr. Galambos is a Distinguished Engineer and certified as an IT architect in the e-Business Architecture Consulting Practice, based in Montreal. In the past six years he has led architecture and design engagements in the finance, insurance, transportation, and government industries. He has contributed as a core team member to the development and deployment of the ESS technical architecture. Prior to his role as a consulting architect, Dr. Galambos focused on IT strategy definition and on the design of high performance/high availability on-line systems and networks for large Canadian and international customers. He used this experience to coauthor IBM's End-to-End System Design Method. Current interests include performance and availability characteristics of the network computing model, the integration server design concept, and asset-based system design. He graduated as a chemical engineer from the Leningrad Technology Institute and received a Ph.D. degree in chemical engineering from the Budapest Technical University in 1972.

Reprint Order No. G321-5697.