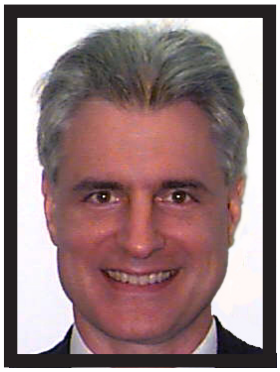
The Siebel Architecture: The Value of a Good Decision

*Presented By:**

William Indest



** A former Siebel employee, Michael Robicheaux, co-authored an earlier version of this paper.*

ORACLE | SIEBEL

Executive Summary

Herein we celebrate the tremendous value from a decision made in the early nineties that has sustained and fulfilled Siebel with enabling technology. The Siebel architecture and the Siebel Data Model are the crown jewels of this company. No other company has produced a software system that sustains 100,000 users concurrently accessing large data stores, support for changes at run-time or tool-time and still have the ability to minimize upgrade pain. We intend to examine the underpinnings of this architectural feat and its ability to support so many different and various types of deployment options, all the while maintaining incredible scalability and performance capabilities and yet remain an easy to customize application. We focus on the issues a CIO and her team of technologists faces, what their view of the world is and what the business drivers and constraints were for us to make such great choices for an enterprise architecture. We describe how our early analysis resulted in an exceptional architecture, the cornerstone of which is the Siebel Declarative Model and we show why it was such a good decision for one of the foundations of the Siebel architecture.

Introduction

In the movie, "Minority Report", Steven Spielberg wanted the visual world to reflect essentially that of which is around us everyday with pieces of the future peeking out. To aid in envisioning this future, Spielberg brought together the persons responsible for helping shape it. Scientists from MIT to urban planners, architects, inventors and writers, the think-tank was created to discuss the social and technological details of our very near future during a three day conference. A near unanimous prediction was the gradual loss of privacy - not so that people can spy on you but so they can SELL to you! In the not-so-distant future, they will keep track of what you buy, so they can keep on selling to you. Well, the future is already here enabled by Siebel's flexible architecture.

But first, let's turn our attention to the issues a CIO faces and those parts of our architecture which address those issues. We zero in on the CIO's pain points so that you can offer the CIO an architectural understanding that supports a process-oriented solution set. We believe that after you read this paper you will appreciate more fully the CIO's perspective and you will understand the value of the good decisions Siebel has made. You should pay particular attention to the list of the more important attributes of the Siebel architecture that we list later. In this paper, we intend to show you how some of those architectural attributes address many of the CIO's concerns; but before we get to that, let's talk about those concerns.

What Are the Issues the CIO Faces?

Let's sit back and look at the problem from a customer's point of view – that of the CIO. There are many issues that may keep a CIO up at night: architecture decisions, technology choices, one platform or multi-vendor, support for standards, complexity of integration, development resources, costs, experience and availability, risks of every kind, efficiency of operations, stability and performance, protection of investment, ability to change, rate of change, global markets, conflicting business goals, vendor trust, vendor relationship, safety of data, options for various deployments, form factor, light weight vs. heavy weight applications and on and on...

Obviously, there are many problems facing organizations trying to convert themselves into a customer driven company in the e-business age. These problems are making CIOs aware of a crucial problem: that their companies NEED an enterprise architecture for their systems whether ERP, CRM and/or SCM. If they haven't already, CIOs are beginning to realize the value of a good architecture choice. CIOs know that unless they plan for redundancy, scalability, application integration and rapid application modification up front, they will be forced to do so later at vastly increased cost. As a result, CIOs are re-embracing the corporate standardized platforms and applications that compose enterprise architecture as a way to contain costs and ensure business alignment in the Internet age.

The CIO needs to create flexible, agile systems. Business requirements are needed to guide the choice of various architectural principles with which the system's flexibility can be attained that is necessary

for evolution. The one constant in business is change. These changes come in the form of market pressure, customer demand, the business units attempting to satisfy those demands, goals set forth by the company leadership, plans being executed daily and finally the pressures imposed by ever changing technology. Systems have to last for a long time, and they have to be adaptable to changing requirements. This dichotomy is one of the nightmares of every CIO, how to make the right decisions to deal with change and satisfy the business in a cost effective way.

> *"[We] need robust IT that allows 'facile integration' of rapidly changing data and organizational transparency so information is shared"* Steven Holtzman, CEO Infinity Pharmaceuticals

The CIO needs to lead her architects to build in attributes that provide for future flexibility and evolution. Future flexibility is an architectural criterion; the value of a "good" architecture compared to other architectures is in its future viability. Modularity, structural stability, and layered solutions are the guiding architectural principles that stimulate the design of the architectural attributes providing for future flexibility.

Why should an organization analyze or worry about architecture? Quite simply, because it is a cost-effective way of mitigating the substantial risks associated with this highly important artifact. Architectures are the blueprints for a system, and the carriers of the system's quality attributes. Enterprise-wide software systems are required to be modifiable and have good performance. They also need to be secure, interoperable, portable, and reliable. But for any particular system, what precisely do these quality attributes - modifiability, security, performance, reliability - mean? For a given business, the CIO may need to delve into a deeper level of some specific attributes more than others.

> *"These days, if you've got Web services you've got dotcom, you've got .Net and J2EE. You need to have some sort of overall architecture blueprint to say where everything fits, the sort of areas you are going to focus on, and the*

sort of technologies you think appropriate for your business and your environment," says Carlo Bonato, manager planning strategies, Victorian Department of Human Services.

Architecture design focuses on "large-scale" concerns while adding constraints and discipline to the design and system implementation. The result is a solution that meets the requirements; not just a solution that simply "works". What are these large-scale architectural issues? Security, distribution, database management, scalability, legacy integration, transaction management, fault management, and application control - all of these should be on the CIO's radar. Some architectural "constraints" are how software components interact, how system capabilities are accessed, and guidelines that are applied to the design of software components. Architecture allows you to decide up-front which of these qualities are most important, and then it is used to constrain design choices. With architecture, the design concepts that most closely reflect the functional objectives can be selected and achieved.

But you may still ask, "Yes, but why is architecture so important?" Unfortunately, without architecture, the chances of having the wrong design are very high. What is the end result? The wrong design means the wrong software solution and the wrong solution means the business's investment has been compromised. Yet, compared to a large software investment, the cost of software architecture is very modest. A commitment to architecture, however, is not easy; it involves significant discipline, process, and skill. The "sphere of influence" that architecture has on an enterprise's projects is large. A sound architecture can significantly reduce project risks, enhance productivity, reduce costs and accelerate time-to-market. Architecture provides a framework of boundaries that imposes rules on software development by providing design and implementation guidelines. Breaking any rule or stepping outside the framework will create a fragile situation and put one or more of the architectural attributes at risk.

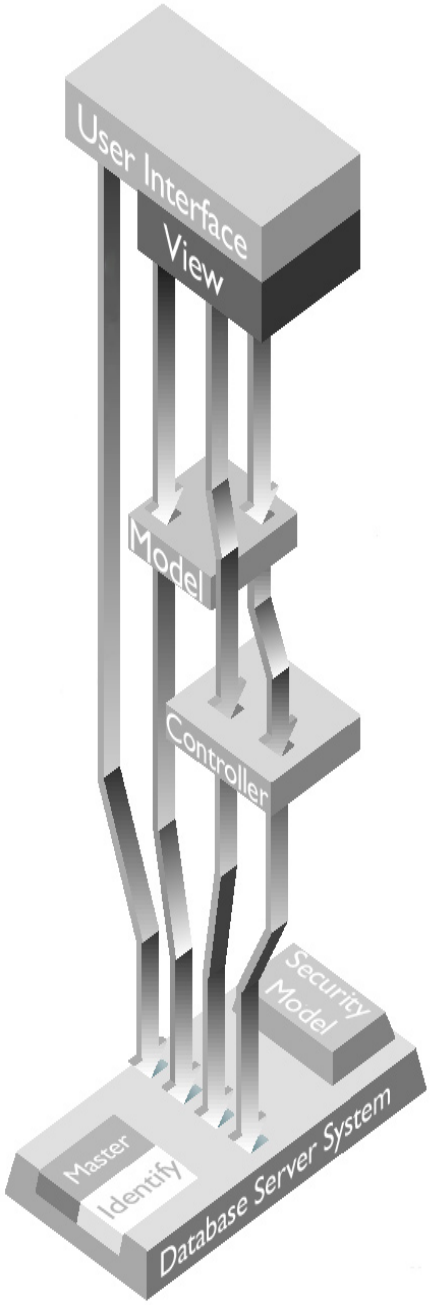
Can a system be analyzed to determine these desired qualities? How do you know if the software architecture for a system is suitable without having to build the system first? This is the

importance of references, especially ones from large corporations, a fact of which our competitors are woefully aware. Experience has shown that the quality attributes of large software systems live principally in the system's software architecture. In such systems the achievement of quality attributes depends more on the overall software architecture than on code-level practices such as language choice, detailed design, algorithms, data structures, testing, and so forth. It is a critical risk mitigation measure to try to determine, before a system is built, whether it will satisfy its desired qualities. But the Siebel system is already built and 1000's of customers are using it so we can skip this step, if you will, and proceed to look at the architectural "tensions".

Availability affects safety. Security affects performance. Everything affects cost. While software implementers and designers know that these tradeoffs exist, there is no codified method for characterizing them and, in particular, characterizing their interactions. For this reason, software architectures are often designed "in the dark". Tradeoffs are made - they must be made if the system is to be built - but they are made in an *ad hoc* fashion. There are some techniques that designers have used to try to mitigate the risks of having to choose the architecture to meet a broad palette of quality attributes.

The recent activity in cataloguing design patterns and architectural styles exemplifies this type of activity. A designer will choose one pattern because it is "good for portability" and another because it is "easily modifiable". But the analysis of patterns doesn't go any deeper than that. A designer using these patterns, does not really know how portable, or modifiable, or robust an architecture is until it has been built. CRM, UAN and UCM are enterprise strategies that require an enterprise-aware architecture as well as an aware and enlightened CIO that recognizes their need. A CIO should understand and know the Siebel architecture. It looms in greatness. It is proven. It is scalable and supports thousands of concurrent users today. And it is upgradeable. This is a great architecture.

Solutions for the CIO's Problem Space



In order to understand how to position and discuss the value of Siebel's solutions with the CIO let us begin with three points we mentioned above – modularity, structural stability and layered solutions. Although inter-related and difficult to discuss without including the other, we try to explain each one independent of the other. The challenge is, modularity exists in a layered architecture AND layered architectures are typically built with modularized components.

The CIO's architecture must be modular. Modularity exists in very fine grained components like business component objects. Modularity is also manifested in large grained components like the various object engines that comprise Siebel's solutions. In the Siebel architecture, modularity is provided by object-based abstractions for software components. Developers can easily reuse any of these objects to specialize them to solve the specific problems for their situation. Risk is dramatically reduced because these base components are used over and over again, proven by all Siebel customers, which builds confidence in the solution. Rapid development is provided by our tools and administration support. Easy access to design time components or easy manipulation of runtime parameters will decrease dramatically the time in which a Siebel solution can be modified or extended. All of these capabilities would not be realized without the architectural foundation supporting modularity. Having modularity reduces the CIO's risk of stovepipe applications appearing in his enterprise systems.

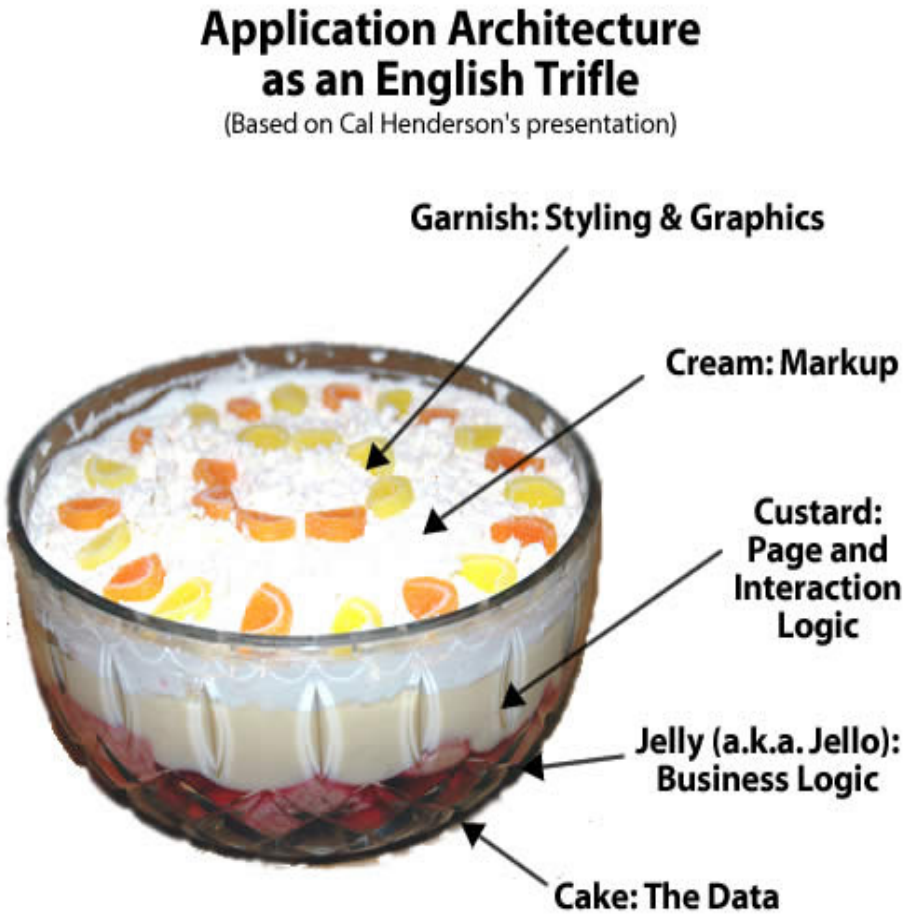
The CIO's architecture must possess structural stability in the form of support for industry standards. We have seen this many times - any number of software design concepts can usually satisfy a project's functional objectives. A software project may be successful even if completed without regard for the overall enterprise and how it interacts with the architecture. These types of projects are how stove piped legacy systems are born. Many designs will "work", but one might be more flexible and pliable than another, or have the modularity needed when making large investments developing an application that will live and grow with the business. The programmer designs the software but the CIO needs to

insure the design of a good architecture before the entrance of the programmer.

An architecture must support layered solutions. The CIO needs to abstract the problems and then separate the solutions of those problems into multiple layers. She then develops a consistent interface between the layers and has the ability to change the implementation without affecting other layers. The challenge is to partition the problem in each layer with the appropriate responsibility of the solution being provided within each layer. Think of the Siebel n-tier layers. The database layer is responsible for fetching data and dynamically converting business object structures into high performance SQL. Such data access responsibilities do not belong in the business object layer. The same holds true for the presentation layer. It is responsible for display and interaction and should not have any business logic functionality. A layered approach leads to heavy reuse which reduces risk while supporting rapid development which reduces cost.

Cal Henderson was the Lead Engineer in building Flickr and has

Yummy Techie Treat!



a wonderful description of a layered architecture using a food analogy that is rather tasteful (couldn't resist). Here is a brief description of each of the layers. What follows is a paraphrasing of his presentation.

"

Software can be divided into layers, each layer having a particular set of responsibilities. The higher a layer is, the closer it is to the user, with the topmost layer being the one responsible for interacting with the user. The lower a layer is, the closer it is to the underlying data and logic for manipulating that data.

Layers are "black boxes" in that their inner workings are hidden from other layers. A layer usually communicates only with the layers immediately above and below it, and only through the interface defined between layers. By "interface", we mean the set of allowed communications between layers.

Garnish

Styling is the cosmetic stuff – it specifies things like fonts, graphics and the like. There is no layer above this one -- rather, it's the user who exists above this layer. The layer below this is the Markup, which get a "look" applied to it by this layer.

Cream

The markup is the user interface which is displayed. In a web application, it's the HTML, in a desktop application, it's the GUI, and to applications accessing this application via the API, it's the XML that is returned by the API calls.

The layer above this one, Styling, applies a "look" to this layer. The user's commands are received by this layer and sent to the layer below -- Page and Interaction Logic. The layer below then provides the responses to those commands, which are then displayed in this layer.

Custard

The page and interaction logic determines how users interact with the business logic. It determines things like which data is displayed to the user or what steps the user must take to add, change or delete data from the system.

This layer receives user commands from the Markup layer above and sends back the appropriate responses. It may need to perform some kind of action that requires fetching data or manipulating it; it does this by accessing the layer below it: the Business Logic layer.

Jelly aka Jello

The business logic is the set of rules which define how data is accessed and how it may be manipulated. The only way the data should be accessed is via the business logic. Cal says that it "defines what is different and unique about the application".

This layer receives user commands from the layer above it; it sends its responses to those commands to that layer as well. Whenever this layer needs to access data, it does so from the layer below it, the Data layer.

Sponge Cake

The data layer forms the underpinning of the application. As with trifle, this layer is big, solid and reliable. All layers above this are "transient and whimsical" (He will always have a home at Oracle.)

This layer responds to requests for data or to change data from the Business Logic layer above it. It sends either the requested data or error or confirmation messages to that layer in response to the requests.

”

Use of a layered solution in our n-tiered architecture combined with modularized components affords the CIO several choices for deployment in order to satisfy the demands of the solutions set, for example, multiple deployment options. A component-based architecture can support a server-based deployment. Remote deployments, where the CIO must push the applications out to the desktop cost more than server-based; this saves a lot of money that would be devoted to remote deployments. Our architecture supports both deployment approaches.

Furthermore, the use of an n-tiered architecture for a system's topology is a critical capability for future flexibility. Our evolution

from client/server to thin-client to web-based high-interactivity client exemplified the capabilities of an n-tier architecture. These architectural changes did not hamper our delivery of a mobile client which is still a market-leading feature of our solution. With this type of layered solution, the CIO can now add layers as technology dictates or as she chooses.

> *"Strategic leverage is shifting to processes that enable an enterprise to sense unanticipated change earlier to enable a proper and timely response...", The Nine Ways in Which CRM Will Change,* **Gartner Report**, January 15, 2004

Here we consider *strategic leverage* as an enterprise-aware architecture that enables the CIO to manifest, dictate and embrace changes: this is how we did it...

Early on we embraced components (back then it was Windows.) If you look at our evolution we embraced an approach to embedding components and changing them declaratively. We created a flexible architecture that is tiered to solve different problems in different layers, while being able to embrace changes quickly. As industry standards became more viable and it made business sense, we began supporting those standards (perhaps even reluctantly). This afforded us the architectural badge of structural stability while proving to be a risk reducer. By embracing standards, Siebel grew to be a much more open architecture, thus giving CIOs several options. They (CIOs) were comforted by the reduction in risk associated with proprietary software components and the ability to choose a different vendor supporting the same standard.

The technology industry constantly changes. Many businesses are re-focusing and they have to realize the benefits of the applications that they have purchased or get rid of them. Companies are no longer buying applications, but solutions. What we need to do as Sales Consultants is convey to the CIO that Siebel has a strong platform that can provide solutions that are component-enabled thus lowering risk and enabling enterprise evolution.

> *"Customer needs-based triggers must*

be linked to specific tactics to build customer value. Some of the more difficult challenges include the need to establish an organizational framework...", The Nine Ways in Which CRM Will Change, **Gartner Report**, January 15, 2004.

We hope the following will help you understand how to position the major attributes of our architecture or *organizational framework* and the value they provide your CIO. While not exhaustive in nature, we believe the essence of our little architecture discussion will provoke more thought and assist you in your ability to help your CIO.

As we described in our previous paper, a declarative development model enables the CIO to respond to changing business requirements quickly, reap the benefits of rapid application development and relax because risks are greatly reduced by the reuse of existing proven components already in the system. This model will support runtime administration or the ability to modify the application WHILE IT IS RUNNING! This provides the CIO the ability to place the appropriate business person in a position to respond to business needs quickly. Concomitantly, IT resources are free to focus on other technical issues. The CIO's team gets kudos due to improved service and satisfied customers, the business people.

Many IT shops do not think about the future because they are focused on the present while maintaining their jobs. As such, they overlook future demands of the system and the needs of the inevitable: migration of systems. Our upgrade capability, a direct result of our declarative model, allows them to leverage their customizations while reaping the functional richness common to a new software release.

In addition, the declarative model lends support to globalization. Globalization eases the evolution to a multi-cultural environment for global enterprises, not an easy thing to do. It definitely is not as easy as it looks. Our architecture supports this capability in an "out-of-the-box" fashion.

Object-based abstractions, a key software development philosophy,

allow an enterprise to test units and reduce overall system testing. The encapsulation of logic and reuse of these logical modules saves time and money and enhances quality. These abstractions led us to develop an n-tiered architecture. Sure when n=2, it is client/server but today our customers have matured and demand an n-tiered architecture. The value of a multiple layered architecture is the ability to run different components on different machines. Doing so allows deployment flexibility, a key requirement for the CIO. Systems can be scaled vertically, a bigger computational box, or horizontally, many smaller boxes devoted to running different components in various levels of the architecture. The decision is driven by the needs of the enterprise and costs.

An architecture based on industry standards or de-facto trends affords many benefits. As standards mature, there is a larger talent pool so the CIO has access to the appropriate resources. As standards are well known, there is shared experience pool of how to use them correctly versus the improper application of the standard (which are valuable pearls of wisdom in their own right). Standards ease integration efforts both internal to the enterprise and externally with business partners. Costs can be reduced and complexity simplified by using standards, thus avoiding the creation of yet another custom interface. Standards-based solutions afford the CIO the option to "unplug" an application and "plug-in" the new best of breed solution.

Our founding fathers, (Tom *et al.*) surveyed customers to aid in eliciting sets of quality requirements along multiple dimensions, analyzed the effects of each requirement in isolation, and then understood the interactions of these requirements. Both processes involve technical and business issues. The technical aspects addressed how the architectural components will envelop not only existing technology but future technology (.NET and J2EE, for example). The business aspects dealt with requirements for solving the customers' problems, providing functional and implementable solutions which are modifiable, flexible and easy to upgrade. Not an easy task to accomplish, but we did.

Many of you are familiar with the Siebel architecture and data model and its ability to change with changing market and customer

requirements. These decisions made over a decade ago, allow Siebel to consistently address market conditions for high performing scalable systems. Some of the more important attributes of the Siebel enterprise application set:

1 - Declarative development model

2 - N-tiered architecture

3 - Use of industry standards or de-facto trends

4 - Object-based abstractions

5 - Runtime administration

6 - Server-based deployment

7 - Globalization

8 - Performance and Scalability

9 - Upgrade capability

10 - Deployment options

The Core Decision: Declarative Development Model

In the beginning our customers told us that some of their main points of pain were upgrading their software system and maintaining their investment. These business requirements drove us to use a declarative development model for the Siebel Enterprise Applications. This approach provides the desired upgrade path while shielding the developer from making coding errors resulting getting the applications to market faster. Siebel Systems can provide to customers several ostensible valuable capabilities: 1) an upgrade path to leverage existing customizations as well as a simple path to the latest version; 2) a design-time development that easily supports an application's hard points; 3) a run-time application environment for modifying the application via a system adminstrator (not a developer); and 4) a configuration application based upon a meta-data repository. These capabilities are highly interrelated and difficult to discuss separately but drive the requirements for the declarative model.

The Declarative Model

The declarative model is a method of development where the developer changes the properties of objects using meta-data in a non-programming environment. What is meta-data? Meta-data is data that describes data. Meta-data is the heart of development based upon a declarative model. Why a meta-data repository? The repository houses descriptions of all objects and their attributes for use by the system. This protects the developer during the development cycle as changes to the value of an attribute are captured in the meta-data repository; while the behaviour of the objects is not changed just the properties. Changing the behavior of an object requires source code access which no Siebel customer has. The repository provides total visibility to any entity in the application and assists in reuse as one can just copy an object, modify it and immediately run the application. In addition, it provides the developer a high-level view of the type of functionality implemented, because all the functionality is stored in one place. As our customers told us that upgrades were costly, time consuming and difficult, simple upgradability became a primary business requirement. Having the information about an application, from the user interface, to the business logic in the object engine to what database and schema to use, reside in a meta-data repository, makes is easier to upgrade from one version to another.

Siebel Tools and our application platform was built from the ground up to support meta-data driven business applications without requiring our customers to write code. Siebel Tools itself runs via meta-data in a configuration application. In fact, Siebel Tools is used to develop Siebel Tools; there is no other tool set. Can a customer write their own code? Absolutely, they pick a language, create a library and they can invoke it from the Siebel application. Remember, flexibility is a required design attribute of a great architecture. Siebel has written the code for the object engines in protected classes that are fully tested by us and functional for all of the requirements of modern business applications. Our customers "configure" Siebel applications by describing their solutions rather than coding to API-based architectures like SAP. The use of standard Siebel components and declarative configuration tools

decreases the amount of time and effort required to customize and maintain the application.

Since meta-data is data, it naturally supports the upgrade process via a merge operation between database instances using relational database technology. This is an elegant and clever way to provide an upgrade path with the flexibility to handle future requirements. The choice of a declarative model gives Siebel's application upgrade path a significant advantage over the competition. One vendor's upgrade process requires a manual **diff** process against source code which is extremely time consuming, difficult, error prone and costly. Many times our competitors' upgrade approach involves a complete reimplementation. The Siebel upgrade path allows our customers to leverage their existing investment in configuring their application to meet their business requirements in addition to taking advantage of added functionality and new technology in our next release.

A huge proof point of the value of the declarative model is when Siebel moved from a client/server to a browser-based application. Customers drove us in that direction to avoid the high cost of application maintenance and installations on the desktop. The Siebel architecture and declarative model allowed the deployment of highly interactive browser-based applications in the enterprise. The declarative model oddly enough can now support .NET and J2EE simultaneously; in other words, declarative development allows the application implementation and execution to be independent of the application definition.

The declarative development model readily supports the architectural re-evolution: client/server; n-tiered; web-client; service oriented architecture. This model insulates Siebel from the details of the implementation technology, allowing the object engines to be implemented in any technology: C++, Java or C#. Siebel's SmartWeb Architecture supports both the thin client and the mobile client, evolving from a client/server architecture to a multiple tiered architecture with incredible flexibility. Amazingly, this Web-based architecture supports both the mobile client as well as the thin client, both running in a web browser. The next step in the evolutionary process is to support a service-based

architecture: .NET and/or J2EE. Can we build it? Yes we can! We have already demonstrated our tremendous flexibility in evolving to different implementation technologies. Today, our customers and technology are driving us to the next evolutionary step.

Summary

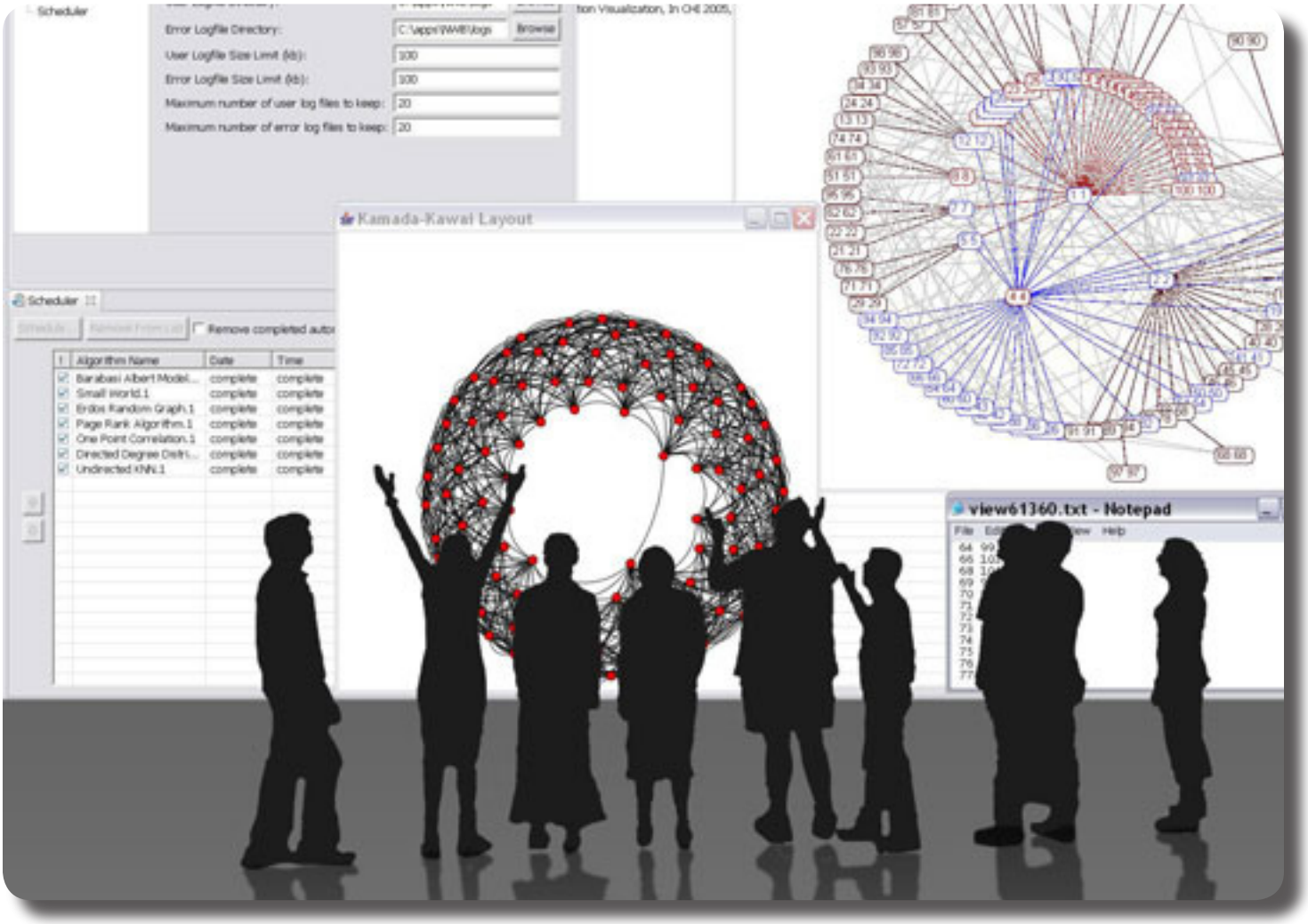
With the declarative model, if you provide a reusable object engine (as Siebel has done), then your application can evolve through many stages: client/server, separated user interface, a web browser based application, and services oriented solutions. Throughout the technology evolution, the declarative model remained a cornerstone for embracing change and appeasing the architectural constraint of flexibility. For example, the future is demanding support for separate runtime environments like J2EE and .NET. Meeting this demand is enabled in part by the declarative development model. As most of us know, Siebel Systems can change directions quickly. This receptiveness and responsiveness is due in part to our decision to use a declarative development model: can you say upgrade easily and prove it with thousands of customers? Hats off to the incredible, talented engineers who made the right decision. What's the value? Many satisfied customers who look to us as a trusted enterprise application partner.

There is clear economic value in using a proven architecture. The use of a well-structured and proven architecture helps new software projects accomplish their goals, produce results that are predictable and reliable and succeed with brilliance. Some software applications have a questionable architectural foundation so it is difficult or often impossible to extend, fix, upgrade or otherwise maintain the resulting software systems. (Can you think of a competitor that has demonstrated the wrong way of designing an architecture?) In other words, paying for a sturdy architecture is well worth its premium price. Unfortunately, not all architectures are equivalent and it's difficult to tell the right one from the wrong one before it's too late. But it can be done. Our jobs as Sales Consultants are to communicate the benefits of the Siebel architecture in the context of the CIO's pain points and help them understand what we provide and how it will dissolve some of their nightmares and provide them a restful night of sleep.

What we have tried to do here is present the CIO's story and what they need to accomplish, not only with a CRM flavor but a perspective into their entire world. They have other problems besides CRM. The "R" in CRM is the key value - RELATIONSHIP! This entire discussion is moot without a relationship with the CIO. What is the value of a good decision? The ability to help our customers solve their problems and build a relationship with them while becoming trusted advisors.

The Future

In Spielberg's "Minority Report" the future envisioned - retinal scanners obtain a positive identification of Tom Siebel as he walks down a mallway; a voice emanates from a speaker asking Tom if he wants to upgrade his current Mercedes to a 1000 series with a set of personally tuned options- is not quite here yet. But the architecture is ready today for this futuristic scenario.



And life is good!

Title: The Siebel Architecture: The Value of a Good Decision
Author: William Indest
The Manager: James Perry

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

EOF