Enterprise Application Integration Encounters Complex Adaptive Systems: A Business Object Perspective

Jeff Sutherland
PatientKeeper, Inc.
20 Guest Street, Suite 500,
Brighton, MA 02135,
United States
email: jeff.sutherland@computer.org

Willem-Jan van den Heuvel
Infolab,
Tilburg University
PO Box 90153
Tilburg, The Netherlands
email: wjheuvel@kub.nl

Abstract

To remain competitive organizations are lining up into virtual alliances, with integrated value chains, introducing competition between, rather than within supply chains. A crucial requirements of virtual alliances, and their supporting, integrated enterprise application is their ablility to move fast and quickly adapt to business-induced change.

This paper combines theories from the area of Complex Adaptive Systems (CASs), that has been successfully applied to explain the adaptive behavior of biological systems, with research from the research arena of enterprise application integration that is based on a variety of distributed business object technologies.

In particular, this paper investigates whether successful EAI implementations conform to the CAS properties by reviewing three case studies that apply business object technology. These concepts can organize our discussion of business object systems and inform our understanding of the success factors for designing integrated enterprise applications, since the most effective integration is often the result of "capturing" software from foreign systems, a common phenomenon in CAS systems.

1 Introduction

The ubiquity of the internet gives enterprises the ability to line up into virtual alliances with tightly integrated value chains, resulting in competition between rather than within vertical industries. It is mandatory that virtual alliances do not only have smootly running integrated business processes, but also have the capability to move fast and quickly adapt to change. Virtual alliances are subject to a wide variety of changes, some of them initiated by themselves, others imposed by external organizations, e.g., changing tax regulations. Business changes must be mapped to the business application level without disrupting the integrated business processes.

To meet the architectural requirements of virtual alliances, and get better reuse from software, distributed business object technology is the preferred solution [4]. Business objects can be the key building block in the integrated company as they can realize domain business processes and default business logic that can be used to start building and integrating business applications in these domains. Furthermore, domain specific models can be designed as business frameworks so they can be easily extended and modified, e.g., SAP and IBM's San Francisco business objects. These can be deployed for integrated cross-enterprise applications that can be easily built upon distributed broker architectures such as CORBA.

One important characteristic of business object technology, that contributes to the critical challenge described above, is the explicit separation of interface and implementation of a class. Business objects technology takes this concept a step further by supporting *interface evolution* in a way that allows the interfaces of classes to evolve without necessarily affecting the clients of the modified class. This is enabled by minimizing the coupling between business components. Client and server classes are not explicitly bound to each other, rather messages are trapped at runtime by a semantic data object that enforces the binding at the level of parameter passing semantics [9]. The key aspect of this mechanism is that messages are *self-defining* as the message name is maintained in the request, and the parameter names are defined in the semantic data object that is also passed in the request.

Currently, distributed object technology, e.g., DCOM, CORBA and Enterprise Javabeans, provide interface description languages and services that allow distributed objects to be defined, located, combined and invoked. The

primary benefit of using them is to encapsulate the heterogeneity of legacy systems and applications within standard, interoperable wrappers. To allow such components to effectively interconnect high-level business objects, business process and workflow applications with low-level legacy wrapper objects, in a way that respects business semantics, new technologies need to be developed. Another major challenge is to permit continuous enhancement and evolution of current massive investments in information sources and systems. In summary, this infrastructure must support the migration of large numbers of independent multi-vendor databases, middleware technologies, and standard packages (e.g., SAP) into *dynamic and highly integrated, evolvable, enterprise information systems* running over distributed information networks. All these are challenges for succesfull integration of enterprise information systems.

Such systems have a highly unpredictable, non-linear behavior where even minor occurences might have major implications. The same phenomen has been observed in the domain of phyiscal and biological systems, be it either an individual animal or a swarm of bees, and has been extensively researched in the area of Complex Adaptive Systems (CAS) [13], [7]. In this article, we investigate the CAS characteristics of succesful EAI projects from a business object technology perspective, so we can derive some design parameters and patterns for successfully integrating enterprise applications.

The remainder of this paper is organized as follows. In the following section, we outline business object component architectures as a stable foundation of enterprise applications, that do not only capture business processes and policies, but also accomodate business-induced changes. In section 3, we clarify the relevance of Complex Adaptive Systems (CASs) to business object systems. These concepts can organize our discussion of business object systems and inform our understanding of enterprise application integration (EAI), since the most effective integration is often the result of ”capturing” software from foreign systems, a common phenomenon in CAS systems. The following three sections serve to illustrate how the CAS requirements within three succesful EAI case studies have been met, resulting in adaptable enterprise integration solutions. Subsequently, section 7 proposes to agentify business objects, that meet the CAS properties, for integrating enterprise applications. This paper is concluded with a summary and some issues for future research.

2 Business Object Component Architectures

Many business object architectures are proposed in literature to support the design and implementation of business object based applications [11]. An architecture defines the stable structure of a system by defining their main parts, their relationships and constraints. A Business Object Component Architecture (BOCA) is an effective solution for dynamic automation of a rapidly evolving business environment [4], as it provides a stable foundation for enterprise applications and accomodates changes to current business practices and policies. Dynamic change requires reuse of chunks of business functionality (as noted by Arthur in his description of the CAS ”software capture” effect [2]). A BOCA must support reusable, loosely coupled, cohesive plug compatible business component so applications can be quickly reconfigured to meet changed business requirements. The two primary strategies now being used for implementing client/server systems to support reengineering of business processes are visual 4th Generation Languages in combination classical distributed object technology. While both of these approaches are recent improvements in system implementation, neither of them effectively implement plug and play business object assemblies.

A group of objects is the ideal unit of reuse. Groups of objects behave as a higher level business process and need a clearly specified business language interface. These so called *business process objects* are a kind of active (or control) objects that bring together business objects to define a business process. They are embodied by a set of interrelated activities that collectively accomplish a specific business objective, possibly, according to a a set of pre-specified policies. Business processes provide the basic ingredients that can be specialized and extended to capture domain or application specific processes – within a particular vertical domain, e.g., financial, manufacturing – which are realized by a workflow. Workflow management systems support the definition, execution and controlling of the business processes. Business processes interact in a predictable, repeatable manner to produce a recognized business activity of generic nature in a specific business domain, e.g., procurement management, general ledger, etc. Business processes are initiated by events that trigger activities in the organization [8]. These events can be internal (e.g., rules) or external (e.g., customer requests). The business processes are initiated on the basis of an incoming event (e.g., a customer request), and result in an outgoing event (e.g., the notification that a product is ordered). So, business processes are concurrently executing, event-driven business objects that are connected to business entities to reflect changes in the business domain.

Proponents of CAS should note carefully Abelson and Sussman’s comments in their classic MIT computer science text [1]. The power of computing lies in recursively writing higher levels of language that are supported by lower level languages (thus inducing emergent behaviors). Layered ar-

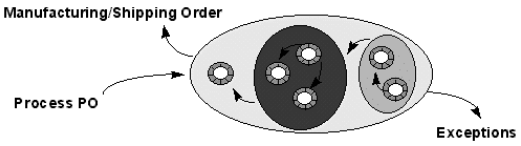

Figure 1. Business Object Assemblies

chitectures, such as BOCA (see previous section), promote discrete recursion as each layer builds on top of lower-level functionality and corresponds to a logical area of concerns. Discrete recursion is considered to be a powerfull concept to reduce complexity and adequately deal with object granularity, as it depends on an architecture with a predefined number of layers, designed for a specific goal [11].

Moreover, business objects need to be encapsulated with a protocol that allows efficient communication with other objects on the network. Work on the concept of Ensembles has shown that there is a minimal design specification for a plug compatible component [18]. A business object needs semantics that extend beyond the syntax of COM components and JavaBeans containers.

More importantly from a business perspective, business objects must guarantee proper behavior in response to a set of scenarios that ultimately trace back to use cases. These scenarios must be evoked by messages in a standardized context, including timing and sequencing, to guarantee proper operation of the component in any system that uses the component according to its design specification. While these minimal requirements were worked out years ago and pursued by the OMG Business Object Domain Task Force since 1995, they have yet to evolve into any standard approach for building plug compatible business objects.

It is critical that standard business objects be available to enable a business system to function as a complex adaptive system. Adaptive means that systems must easily evolve and this requires continuous changing, updating, adding, subtracting, and rearranging components. When changing one component causes ripple effects throughout a system, the cost of rewriting major portions of the system slows evolution to a snails pace. This is the state of the 80% of business systems that are viewed as unsuccessful implementations in industry today [5].

Several architectures to structure the tiers of business objects have been proposed in literature: the Business Object Component Architecture (BOCA) [23], and, the layered architecture as proposed by Prins [28], and lastly, the User-Workspace-Enterprise-Resource (UWER) architecture [11]. However, these architectures do not conform to essential CAS characteristics, such as nonlinearity and mechanisms that contribute to adaptability. These characteristics are discussed in the next section.

3 Complex Adaptive Systems (CASs) from a Business Object Perspective

Holland defines CAS as systems composed of interacting agents which respond to stimuli and stimulus-response behavior that can be defined in terms of rules. Agents adapt by changing their rules as experience accumulates and can be aggregated into meta-agents whose behavior may be emergent, i.e. not determinable by analysis of lower level agents [13].

Business entities are good examples of complex adaptive systems. The modification time of a business firm is on the order of months or years, about the same amount of time required to enhance its computing systems. Automating business processes renders a subset of the business in software, thus enterprise software systems are examples of CAS. A business system has a severely constrained rule set, compared to a typical CAS system like New York City or the U.S. economy, making them ideal for initial implementation of CAS concepts. Business benefits can be significant since enhanced flexibility, adaptability, and reusability of these systems can strengthen the ability of an enterprise to evolve and survive in the marketplace. Typical characteristics of CAS systems are lacking in most business software systems so there are major opportunities for improvement. New discoveries in object technology parallel Holland's analysis of CAS. Event driven, distributed object component systems are "interactive systems."

Wegner has shown that interactive systems are not Turing machines [33]. All interactions in these systems cannot be anticipated because behavior emerges from interaction of system components with the external environment. Such systems can never be fully tested, nor can they be fully specified. These contemporary, event driven, business object component systems exhibit emergent behavior, a fundamental feature of CAS. And they are clearly complex, to the extent that the running system may be the shortest description of its behavior. This is certainly true of large enterprise systems running in multiple sites with different custom code and hardware platforms at every site. Supply chain integration of these disparate systems via an intranet, extranet, and/or the internet is often a global EAI development task.

Business object components can be defined recursively, and represented at various levels of granularity. Objects are aggregated into components. Components are aggregated into meta-components and a hierarchical structure is built to support integration of enterprise software systems. Holland uses an equivalent diagram to describe a complex adaptive

system made of of adaptive agents (see Figure 1.1, [13], p. 6). Work on Business Object Component systems is evolving along the lines of CAS, because successful systems are, in fact, instances of complex adaptive systems. Of the 80% of all software projects deemed failures, reasons for failure are often related to the inability of systems to rapidly adapt to changing business needs [5]. In fact, the failure to design software systems to support CAS behavior dooms the majority of them to failure and the minority survivors to premature obsolescence because of their inability to adapt through integration with newer system developments.

Enterprise application integration from a Business Object Component/CAS perspective requires an understanding of Holland's synthesis of CAS concepts:

1. **Aggregation (property)** - there are two important modes of aggregation in CAS systems. Aggregation is a basic mechanism in object modeling and is the basis for identity, a fundamental object concept. Forming components out of objects and enterprise systems from components is higher level aggregation. More important are emergent properties such as intelligence that evolve out of dumb subsystems. This is the basic concept in Minsky's "Science of Mind" [20] or Hofstader's analysis of an ant colony [12]. Meta-agents (an enterprise) are formed of aggregates of agents (enterprise systems) and exhibit emergent behaviors (revenue, profitability, and cash flow, the indices of value creation).

2. **Tagging (mechanism)** - this mechanism facilitates the forming of aggregates, from HTML pages to the mechanisms in CORBA or DCOM that allow inter object communication. They facilitate selective mating, i.e. firewalls block certain tagged elements to protect the enterprise. Thus they preserve boundaries between aggregates. They allow us to componentize object models and enable filtering, specialization, and cooperation. They are the mechanism behind the development of hierarchical aggregates that exhibit emergent behaviors like an operating system. The basic mechanisms of evoking operations through messages in object technology are based on tagging strategies.

3. **Nonlinearity (property)** - nonlinear systems exhibit catastrophic and chaotic behaviors. Traffic flow on the Internet is nonlinear, leading to predictions of the collapse of the network. Brownouts, system loadings, scalability effects are often nonlinear. The arrival, proliferation, and destruction of viruses on the Internet is a nonlinear phenomenon that can be modeled like predator/prey interactions in biological systems. Even more basic phenomenon like revenue prediction in an enterprise financial system has nonlinear behaviors that are often not recognized. The rate of construction of software itself is a nonlinear phenomenon.

4. **Flows (property)** - workflows are examples of flows in action. Message routing is a flow. Tags condition flows which often exhibit nonlinear characteristics and emergent behaviors. Flows typically have a multiplier effect. Money injected into the economy has an effect out of proportion to the amount, similar to email or other message flows on a network. The recycling effect of flows enables the rain forest, as well as an enterprise computer ecosystem. Individual pieces evolve, die, are replaced or reused, constantly changing the characteristics of the enterprise. Living software is software that is constantly changing due to flows, as rivers change their course. Dead software is eventually detritus that is expelled from the enterprise organism.

5. **Diversity (property)** - persistence of an individual agent depends on the ecosystem of agents that surround it, whether the agent is an ant in the rain forest or a business object in an accounting system. The evolution of these agents as software changes causes convergence of system architectures. This is the basis of emergent patterns that reappear again and again in widely disparate environments. It is difficult to evolve a single agent to make it more useful in an isolated context. Usefulness in business object systems arises from interactions between diverse agents as in human societies.

6. **Internal models (mechanism)** - the utility of complex systems is enhanced if the system can learn from experience and adapt its behavior. The ability of the system to develop and act on internal models that simplify the external world is basic to this mechanism. It allows the system to infer the results of actions before they are taken, and to choose actions that have productive results. The prospects for longevity of software systems depend on this capability, just as in living systems.

7. **Building blocks (mechanism)** - reuse is dependent on building blocks used over and over again. It is the basis of Moore's law in hardware production. It could be the basis of dramatic improvements in software productivity. Building blocks are the basis for generation of internal models and are essential to the construction of adaptive enterprise systems.

Holland, in his Ulm lectures at the Santa Fe Institute creates a synthesis of these seven basic CAS concepts, four properties (aggregation, nonlinearity, flows, diversity) and three mechanisms (tags, internal models, buildings blocks) [13].

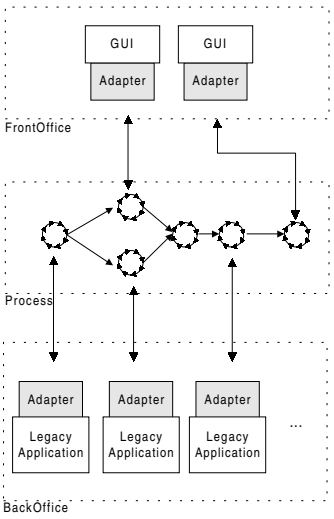

Figure 2. Integrating two disparate systems [32]

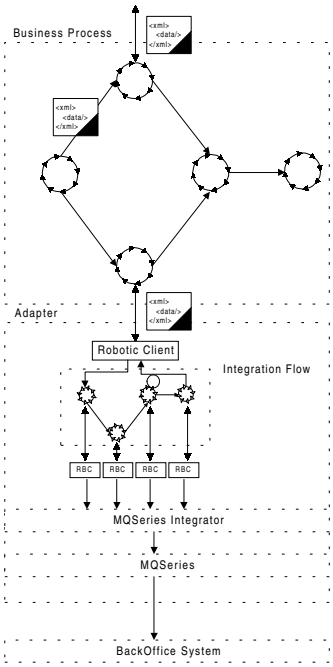

Figure 3. Overview of the backoffice adapter [32]

4 Case 1: The Merger of two Insurance Companies

Van den Enden et al. [32] provided a creative example of ”capturing” two enterprise systems. The merger of two insurance companies required disparate system integration. The front office system from the first company was built on Sun hardware using the Forte 4GL environment. The second company had a backend system that ran on Unisys mainframe hardware and was coded in LINC, a Unisys code generation language.

Van den Enden et al. decided to use a workflow engine, intelligent adapters, and XML messaging [3] as the core of their integration strategy. New technology was superimposed on the old in order to enable (1) interaction between web HTML clients and mobile WML clients, (2) utilization of standard transform mechanisms with XSL/XSLT, (3) easy integration with future systems in electronic marketplaces through XML, and (4) validation and language mapping capabilities available with XML DTD and XML Schema tools. Van den Enden’s architecture ”captured” the disparate systems with a workflow engine (Sun’s Forte Conductor). All business logic is encapsulated in a workflow, and the architecture uses intelligent adapters to provide for the ’glue’ that links the external applications to the workflow. Adapters transform XML message formats into other data formats or into objects and are able to take different actions based on the content of the message.

Sun’s Forte Conductor graphical tools are used to set up a process definition which describes how to initiate a process, what step the process is in, and who is responsible for executing the step (the front end user or the backend system). All details of the frontend or backend systems are hidden from the workflow layer. Thus Forte Conductor ”conducts” the execution of a process instance by human and machine interactions.

The backend architecture of this system is depicted in Figure-2. When a node in the process instance requires the cooperation of the backend system, it sends an XML message to a ”robotic” client. This agent orchestrates the flow of commands required to integrate the backend system into the workflow. Similarly, the Conductor workflow engine sends XML messages to a frontoffice adapter which insulates the frontoffice system from the workflow layer.

In this architecture, two legacy systems are aggregated into a single system by adapters controlled by the workflow engine. Nonlinear behavior induced by actions of goal seeking agents is avoided in this system by essentially hardcoding workflows and tuning them prior to production. Flows are managed by the workflow engine and routed using tags supported by the infrastructure of XML. and diversity is managed by shielding the workflow engine with ”robotic” clients. Internal models are essentially hardcoded. And building blocks used in aggregation are enabled by workflows interacting with adaptors that manipu-

late legacy systems. We would not expect emergent behavior in this system because of hardcoding workflows and implicitly defined and fixed internal models. However, since the architecture abstracts business processes from the more rigid legacy system, it would support future extension of the system to support agents with adaptive internal models that could dynamically define workflows within the limits of defined adapters. In particular, this system implements a fundamental concept that will be characteristic of future adaptive systems - automated workflow drives business processes, in place of human interaction. This is an excellent example of "capturing" the software of two legacy systems and repurposing them for future flexibility. It preserves legacy investment while freeing legacy systems from many limitations.

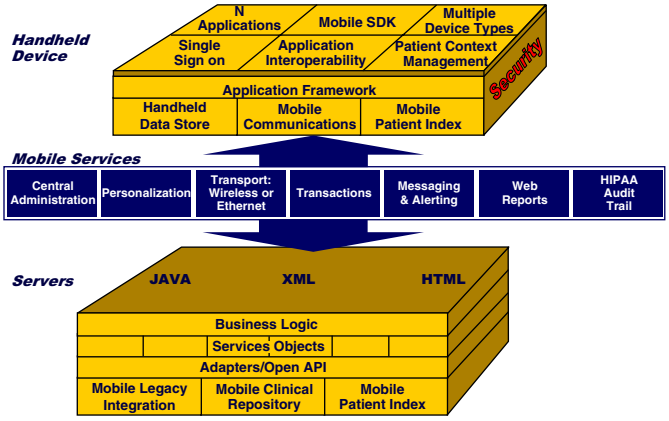

Figure 4. The PatientKeeper Architecture

5 Case 2: Mobile Device Platform support for the Integration of Hospital (Legacy) Information Systems

The Internet explosion has inspired an outpouring of predictions that the health sector's long-awaited breakthrough in information management is finally at hand. But will network computing really help create order amid the befuddling maze of insurance claims, clinical records, and quality data in which the key to a more efficient system now lies hidden? Or will the ultimately localized, idiosyncratic, and fragmented enterprise of care continue to prove resistant to rationalization?

A more sophisticated problem of enterprise integration is introducing "point of sale" technologies into healthcare. Most people are unaware that the United States is tied with Croatia for the lowest level of automation at the point of care of any country on the planet. Over 95% of clinicians have no automation at the point of care in the United States versus more than 90% with some level of automation at the point of care in the United Kingdom and Canada [6].

The net result is a tremendous loss of money due to unbilled or incorrectly code charges (an average of 8-10% in large Integrated Healthcare Delivery Networks). Loss of funds in failed reimbursement for laboratory tests due to miscoding or failure to demonstrate medical necessity can result in the lost of tens of millions of dollars annually for a single institution. Even worse is the fact that medication error is (conservatively) the fourth leading cause of death in the United States and that minimal levels of automation would reduce these deaths by 50-80%. This has been declared a national emergency by the National Academy Insitute of Medicine [16]. In fact, when all sources of iatrogenic deaths are included, such as nosocomial infections, preventable medical error is the third leading cause of death by a wide margin, exceeded only by heart disease and cancer [30].

Solving this problem requires a complex enterprise application integration effort. First, a large Integrated Delivery Network (IDN) will often have hundreds of disparate software systems, several of which will need to be integrated to support even a single limited application like generating a bill for treatment. Second, physicians are inherently mobile and have no immediate access to any of the information in these hundreds of systems. Adoption of personal computers and online medical records is vanishingly small at the point of care. However, almost 30% of physicians today carry a mobile device such as a Palm Pilot and the number is growing quickly due to physician interest. Third, the cost of mobile devices and wireless communication is dropping rapidly to the point where total cost of ownership (TCO) is 1/5 the cost of supporting a laptop. One could argue that reducing the fourth leading cause of death by 50-80% is the most humane act that computer professionals could possibly provide in terms of reduction of human pain and suffering [21].

Tackling this problem requires four layers of distributed systems technologies. Cache consistency across all layers of these systems must be maintained precisely to avoid medical error [10]; [17]. An integration platform architecture for support of mobile/wireless applications at the point of care is shown in Figure-4

The top layer of this architecture is the mobile device. While the platform needs to support a wide variety of device types, an individual clinical only wants one device in their coat pocket. The handheld device provides a mobile patient index integrated with backend legacy systems and an application API which allows independently authored applications to plug into the framework. All applications are inter-

operable in the sense that the clinician selects a patient, then selects an application which automatically understands the patient context. Key requirements are a single signon and security infrastructure, application interoperability, and "always ready" operations. When the device is disconnected from the network, applications run normally off the local device datastore. When the device is reconnected by any wireless or wired mode, it automatically synchronizes with backend databases and runs as a connected application.

The second layer of the architecture is a synchronization server which must support a wide variety of device types. Information flowing to the mobile device must be personalized to the specific device, clinician, and application. For example, if Dr. Palm is a cardiologist, he has a set of applications oriented towards cardiology. He wants to see his own patients on his Palm Pilot and their lab results. If he is in his office, he may want to be alerted on his Palm Pilot. When he is on the golf course he may want to be paged. If he is on call he will want another physicians patients to transparently appear on his mobile device. The synchronization server must stage data, manage personalization, and handle routing, alerting, and messaging. It must keep the data cache on the mobile device synchronized with lower level databases in the architecture.

In order to manage a complex set of clinical and financial data requiring extensive authorization, security, and auditing features, a robust, fault-tolerant, clinical repository is required. This is the third layer of the architecture. In addition to managing patient data, the repository must manage multiple record numbers for the same patient which exist independently on dozens of legacy systems. It must also manage a complex network of enterprise application integration varying between dozens of different applications. For example, if Dr. Palm is in the hospital on rounds he wants data to flow to and from the hospital financial and clinical system. When he goes back to his ambulatory clinic, he wants to see data flowing to and from his clinic financial and clinical systems. He wants all this to be transparent to whichever hospital or clinic he is in at the moment. He wants data seemlessly available anywhere or he will not use the device.

The clinical repository in this example is integrated using a component object strategy. The repository provides object-oriented component interfaces to the synchronization server. The messaging protocol between the synchronization server and the repository is XML using SOAP [22] as an RPC mechanism. The critical need for a component architecture to provide seamless integration is discussed in section 2.

The fourth layer of the architecture are the legacy systems themselves. They can be integrated via adapters as in the previous insurance example, or interfaced using standard message formats or proprietary legacy interfaces.

The architecture aggregates multiple heterogeneous systems through component objects, adapters, or messaging interfaces. It also aggregates data from backend legacy systems and manages the consistency of data across layers of the architecture. Flows are handled by workflow engines at both the synchronization server and clinical repository layers of the architecture. Tagging is supported by XML infrastructure. Diversity is shielded from the mobile device by the clinical repository. Building blocks are aggregated with a wide variety of integration patterns and mechanisms for interoperability. The system is complex enough to induce nonlinear behavior but this must be managed by manual tuning or coding. Internal models are implicitly defined by hardwiring components for specific applications. There may be limited goal seeking behavior induced by hardcoded business rules in various system layers.

A unique feature of this architecture is that mobile devices can be used as a remote control to drive the enterprise. Workflow drives business processes. However, humans at the remote control serve as intelligent agents with goal-seeking behaviors. This architecture is an excellent attempt to handle complexity but is not adaptive. New configurations must be manually created. To raise the level of adaptability requires moving a workflow engine to the center of the architecture while simultaneously distributing it across all computing platforms in the enterprise. This is the focus of the third example, but first some comments on component architectures essential to the workflow strategy.

6 Case 3: "Big Workflow" for Enterprise Application across the Healthcare Integrated Delivery Network (IDN)

"Big Workflow" is a term that has been used to describe workflow that crosses multiple applications and multiple vendors to support patient flow across a healthcare Integrated Delivery Network (IDN). Similar systems have been implemented or are under construction in manufacturing and other vertical application domains. As an example, HealthSystems Minnesota has carefully defined their business processes for managing patient flow across multiple institutions within their IDN. They would like the capability to superimpose HealthSystems business processes across all (more than 300) vendor applications, and be able to modify them globally without modifying vendor systems.

Sutherland and Alpert presented an implementation of "Big Workflow", in [31]. This work was initiated based on research by Santanu as presented in: [26]. He describes a

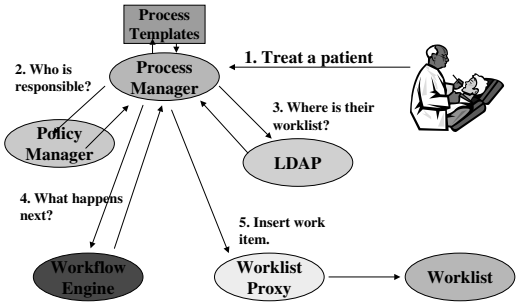

Figure 5. Internet Workflow for the Enterprise

review of work on Rainman and pointed out deficiencies in the WfMC architecture [26]. More in particular, this standard is not well-equipped for managing workflows over the Internet as the workflow server is monolithic, worklists are hidden within the workflow server, and the connection between clients and the server is synchronous [25]. This yields an inflexible and non-scalable solution.

"Big Workflow" was implemented based on several fundamental requirements. Any server on the Internet can host any component of the system (process manager, workflow engine, observers, worklists). W3C standards (HTTP/XML/SOAP/LDAP, see http://www.w3c.org/xml) was used exclusively in the implementation. There was no single point of failure, and any system that supports an XML remote procedure call (SOAP) could participate in the workflow.

Workflow can be viewed as a primary component in architecting applications where flow of control is abstracted out of application business logic. The application model layer (see figure below) becomes a set of services that can be manipulated by a workflow engine. Alternatively, a legacy system can expose a set of services that can be driven by an external workflow engine. If the user can create and alter process definitions (workflow protocols), a process definition can serve as a simple agent which drives business processes.

The workflow system is composed of a process manager that receives an event. The process manager then looks in the LDAP for a URL for the appropriate process template, policy manager, and workflow engine. The process manager using the workflow engine to determine the next step in the process, queries the policy manager to see who is responsible for that step (human or machine), queries the LDAP for the URL of the worklist for the responsible party, and posts the work item on the appropriate worklist somewhere on the Internet (see Figure-5).

The system can set up multiple Observers to view the state of any workflow anywhere on the Internet. A Worklist will notify participating Observers upon any change in state. A distinguishing feature of this implementation is that all workflow context is distributed to worklists. There is no central point of control. Any Workflow Engine on an server on the Internet can drive the workflow process. If an Observer notices that a workflow process is not completing in an appropriate period of time, the Observer can delegate this work item to another server. Thus the system as a whole is fault tolerant and scalable with the capability of automatically routing around bottlenecks.

This system was the first fully distributed Internet workflow implementation that was sufficiently scalable, fault-tolerant, and about to reroute around bottlenecks or failures known to the OOPSLA'99 Workflow Workshop participants. In the view of IBM T.J. Watson Laboratory scientists, it transcended anything created at IBM. Any application on the Internet that exposed a set of services with an XML/SOAP interface could participate in any workflow. Such applications could be automated to interoperate on a global scale. This was viewed as a basic platform on which to build the next higher layer of an architecture that could support goal-seeking CAS behaviors. This is a major thrust of future research directions.

7 Agententifying Business Objects

Engineers working on Business Object Component Architectures and EAI would benefit by gaining an understanding of CAS concepts and applying them to their work. These concepts are larger in scope and broader in application than some of the Business Object work now under development [11], [4]. Some of the best logical and mathematical minds of our time are gravitating to CAS research. We can expect fundamental breakthroughs in our understanding of how to build complex adaptive systems that could translate directly to Business Object Component Architectures if the dialogue and definitions used by Business Object specialists aligned with CAS research. Furthermore, the Business Object community could provide some of the best testing ground for CAS concepts and fill in the major gap in CAS research by providing hard emperical data on production EAI systems exhibiting CAS qualities.

Interactive, autonomous, business object components could evolve independently with a full implementation of an agent-based enterprise system. Some of these components could be intelligent agents roaming the net to perform complex tasks based on enterprise workflows. Workflows in these systems must be managed in a more sophisticated way than current Workflow Coalition or the Object Management Group specifications prescribe [29]. They will ex-

hibit complex behaviors, catastrophic events, and chaotic interactions, all phenomena subsumed under the umbrella of ”complex adaptive systems (CAS).” They are under intensive research for use in predictive economic models (let the computer beat the stock market, deploy new derivatives, or perform arbitrage), the building of artificial life forms for the analysis of biological systems, computer models that can independently adapt and evolve, ”avatars” that can personally represent the creator in an Internet chat room, to note only a few examples.

The previous examples had static implementations that did not allow the system to adapt its operations based on the dynamic state of the runtime environment. Next generation systems will allow business object components to decide with whom to collaborate, what services to offer, what services to request, and what visible behaviors to exhibit [19]. Business object (BO) components need to be “agentified” using a multilayered architecture.

Coordination between BOs is carried out by ”conversations” based on well-known coordination strategies [15]. A Business Object Communication Language (BOCL) needs to be standardized as part of a Business Object Component Architecture (BOCA) to provide consistent interoperablility between business object component systems. Some work in this area has already been done, e.g., the development of a special version of the Formal Language for Business Communication (FLBC) [34], that is capable to formally describe communicational business patterns between two or more intelligent agents. Assuming that operations in a BO are packaged into workflows that can be driven automatically by a ”Big Workflow” implementation, a BO could dynamically aggregate BO building blocks to execute goal seeking behavior.

These intelligent agents constitute the next higher level of abstraction to business object technology, and design on the basis of CAS characteristics, are capable of not only modeling and implementing business processes based on “natural” business object technology, but at the same time negotiating with other agents about their (mutual) goals [27], and dealing with change a more pro-active way than ever before.

8 Conclusions and Future Research

Business object components that evolve over time are complex systems. If progamming is required, they evolve slowly. If adaptation is automated, they evolve quickly. By viewing business object systems through the ”lens” of CAS we can better assess their flexibility and adaptability and evaluate their architecture on a scale of poor to excellent.

The three case studies presented show (1) the integration of legacy systems through ”capturing” the legacy systems by wrapping them with a new layer of technology, (2) the ”subsumption” of legacy systems by layering them below a new generation of technology, and (3) integrating advanced concepts is distributed workflow into the subsumption architecture to provide a framework for agent based evolution of business object systems.

The future direction of research is to integrate agent-based, conversational business objects into a workflow framework to produce a CAS system that can evolve with minimal human intervention. New components should be capabable of being introduced into the architecture and detect how to plug and play with previously existing components. In this way, we could have a software architecture that could evolve as quickly as business processes are now evolving in 21st century organizations.

References

[1] Abelson, H., Sussman, G.J., et al. Structure and interpretation of computer programs. Cambridge, MA, MIT Press, 1996

[2] Arthur, W. B. (1994). On the Evolution of Complexity. Complexity: Metaphors, Models, and Reality. Proceedings Volume XIX, Sante Fe Institute Studies in the Science of Complexity. G. A. Cowan, D. Pines and D. Meltzer (eds), Addison-Wesley, 1994.

[3] Bosak, J. XML, Java, and the future of the Web. Sun Microsystems, 1997.

[4] M.L. Brodie. The emperor’s clothes are object oriented and distributed. In M.P. Papazoglou and G. Schlageter, editors, *Cooperative Information Systems: Trends and Directions*, pages 15–48. Academic Press, 1998.

[5] Brown, W. J. . AntiPatterns: refactoring software, architectures, and projects in crisis. New York, Wiley, 1998.

[6] Bushko, R. (1997). Conference overview: Chairperson’s remarks. Future of Health Technology, MIT Media Laboratory, Cambridge, MA, FHTI.

[7] Cowan, G. A., D. Pines, et al. Complexity : metaphors, models, and reality. Reading, Mass., Addison-Wesley, 1994.

[8] Curran, T. and Ladd, A. (2000). *SAP R/3 Business Blueprint: Understanding Enterprise Supply Chain Management*. Prentice Hall, Upper Saddle River, 2nd edition.

[9] P. Eeles and O. Sims. *Building Business Objects*. John Wiley & Sons, New York, 1998.

[10] Franklin, M. J., M. J. Carey, et al. Transactional Client-Server Cache Consistency: Alternatives and Performance. ACM Transactions on Database Systems 22(3): 315-363, 1997

[11] Herzum, P. and Sims, O. Business Component Factory John Wiley & Sons, New York, 2000

[12] Hofstadter, D. R. Gödel, Escher, Bach : an eternal golden braid. New York, Basic Books, 1979

[13] Holland, J. H. Hidden order : how adaptation builds complexity. Reading, Mass., Addison-Wesley, 1995.

[14] Inglehart, J. K. The Internet and Health: Vision. Health Affairs 19(5): 8, 2000.

[15] Jennings, N., K. Sycara, et al. A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems, 1(1): 7-38, 1998.

[16] Kohn, L. T., J. Corrigan, et al. To err is human : building a safer health system. Washington, D.C., National Academy Press, 2000.

[17] Lee, S., C.-S. Hwang, et al. Supporting transactional cache consistency in mobile database applications. Proceedings of the ACM international workshop on Data engineering for wireless and mobile access, Seattle, ACM, 1999.

[18] Love, T. Object Lessons: Lessons in Object-Oriented Development Projects. New York, SIGS Publications, 2000.

[19] Maamar, Z. and J. Sutherland. Toward Intelligent Business Objects: Focusing on techniques to enhance BOs that exhibit goal-oriented behaviors. Communications of the ACM 43(10): 99-101.

[20] Minsky, M. L. The society of mind. New York, Simon & Schuster, 1998.

[21] Napoleone, D. and M. Berger. Physician Order Entry Implementation at Montefiore Medical Center. TEPR 2000, San Francisco, CA, TEPR., 2000

[22] Nielson, H. Introduction to SOAP. OOPSLA XML Process and Objects Symposium. Proceedings of the 15th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, Minneapolis, ACM.

[23] Object Management Group. Common facilities RFP-4: Common business objects and business object facility. Technical Report OMG TC Document CF/96-01-04, Object Management Group.

[24] Paul, S., E. Park, et al. RainMan: A Workflow System for the Internet. Technical Report. IBM T.J. Watson Research Center, 1997.

[25] Santanu Paul, Edwin Park, Jarir K. Chaar: RainMan: A Workflow System for the Internet. Symposium on Internet Technologies and Systems, Monterey, California December 8-11, 1997

[26] Paul, S., E. Park, et al. Essential Requirements for a Workflow Standard. In: Business Object Design and Implementation II: OOPSLA'96, OOPSLA'97, and OOPSLA'98 Proceedings. D. Patel, J. Sutherland and J. Miller (eds). London, Springer-Verlag: 100-108, 1998.

[27] Papazoglou, M. Agent Oriented Technology in Support of E-Business: Enabling the Development of "Intelligent" Business Agents for Adaptive, Reusable Software Communications of the ACM, 44(4):71-77

[28] Prins, R. *Developing Business Objects: A Framework Driven Approach*. McGraw-Hill Publishing Company.

[29] Schmidt, M.-T. Building Workflow Business Objects. In: Business Objects Design and Implementation II: OOPSLA'96, OOPSLA'97 and OOPSLA'98 Workshop Proceedings. D. Patel, J. Sutherland and J. Miller (eds). London, Springer-Verlag, 1998.

[30] Starfield, B. Is US Health Really the Best in the World? JAMA 284(4): 483-485, 2000.

[31] Sutherland, J. and S. Alpert Big Workflow for Enterprise Applications. In: OOPSLA Workshop on the Implementation and Application of Object-Oriented Workflow Management Systems II., Denver, CO.

[32] Van den Enden, S., E. Van Hoeymissen, et al. A Case Study in Application Integration. In: OOPSLA Business Object and Component Workshop. 15th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, Minneapolis.

[33] Wegner, P. Interactive Foundations of Object-Based Programming. IEEE Computer 28(10): 70-72, 1995.

[34] H. Weigand and W.-J. van den Heuvel, Meta-Patterns for Electronic Commerce Transactions based on the Formal Language for Business Communication (FLBC), International Journal of Electronic Commerce, 3(2): 45-66, M.E. Sharp, Inc, 1999