By Jeff Sutherland and Willem-Jan van den Heuvel

Enterprise Application Integration and COMPLEX ADAPTIVE SYSTEMS

Could system integration and cooperation be improved with agentified enterprise components?

The ubiquity of the Internet enables enterprises to line up into virtual alliances with loosely coupled business processes organized along the axis of the virtual value chain, resulting in competition between rather than within vertical industries. This highly challenging process has recently evolved through several levels of integration from isolated legacy silos of enterprise information to agile and cooperative information systems. Batch data transfers have traditionally been accomplished through nightly magnetic tape inputs in banking systems. These have evolved into real-time point-to-point interfaces extensively used in banking, health care, and manufacturing systems to coordinate data flow between systems. This traditional Enterprise Application Integration (EAI) is tedious, expensive, and inflexible.

More recently, business object components have been used to provide a services-oriented interface to external systems and allow interoperability between systems. Tightly coupled approaches such as COM and CORBA have proven inflexible in an inter-enterprise context. Business-to-business Web transactions are becoming loosely coupled through XML/SOAP interactions. In addition, coordination of activities to provide business process support between business partners is enabled through workflow architectures driving loosely coupled systems. Finally, services on demand requires response to goal-seeking behaviors exhibited by software agents, the ability to search for compatible components and services, methods for establishing contracts and commitments between alien agencies, and marshalling binding between disparate systems.

The preceding phases of EAI exemplify the fact that progressive organizations are eager to move from process integration to support long-term business relationships with more volatile organizational shapes, such as flexible virtual alliances, with loosely coupled business processes and supporting Web-services orchestrations that are in fact *agentified* and exhibit Complex Adaptive System (CAS)-like behavior. Virtual alliances are subject to a wide variety of (un)expected changes; some of them initiated internally (reinvented business policies), others imposed by external organizations (changing tax regulations). Despite the fact that it is not possible to anticipate all consequences of such business changes, they must be swiftly mapped to the business application level without disrupting integrated business processes.

Improvements in productivity are required to fulfill these critical requirements. In the 1990s, the growth of productivity was assumed to be caused by more effective development and use of IT, as opposed to the 1980s when no increase in productivity from computing was visible (Solow's paradox). Closer examination reveals that recent increases in output have occurred primarily within the semiconductor and related industries (Moore's Law effect). Productivity has actually declined in many industries with large software implementations such as hotels, retail banking, long-distance communications, and health care. This may be due to widespread implementation of stovepipe technologies acting as isolated islands of automation that cannot collectively adapt to changing business needs.

Our premise here is that the economic benefits of computing rely heavily on integration of disparate systems as opposed to traditional point-to-point interfaces; the capture and control of legacy technology in a way that forces alignment with business change (even in real time); and

system evolution that enhances, rather than disrupts, output. Evolution of enterprise systems in real time requires new approaches to systems integration and adoption of CAS techniques to manage complexity while providing flexibility and adaptive behavior.

Moving Beyond Point-to-Point Interfaces

To meet integration and adaptability requirements and accomplish more effective reuse from existing business applications, distributed business object technology is generally perceived as the ideal solution. Business objects can be aggregated into enterprise components that provide business services. Enterprise components can be key building blocks in the integrated company as event-driven business processes are realized. Moreover, they facilitate the refactoring of a legacy system by wrapping them and integrating them in new applications. Assembling enterprise components into domain-specific applications can be greatly leveraged by organizing them into business frameworks, which can be easily configured and deployed while relying upon distributed broker architectures such as CORBA [4] or emerging Web services standards such as the W3C Web Services Description Language (WSDL) [2].

One important characteristic of business object technology, which contributes to the critical challenges described previously, is the explicit separation of interface and implementation of a class. Enterprise component technology, in aggregating business objects as services, 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 enterprise components and applying self-describing messages to describe the parameter passing semantics [3].

Currently, distributed enterprise component technology, such as Enterprise JavaBeans and .NET, 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. This infrastructure must support the migration of large numbers of independent multi-vendor databases, middleware technologies, and standard packages (IBM's WebSphere and MySAP) into dynamic and highly integrated, evolvable, enterprise information systems (EISs) running over distributed information networks. Thus, integrated enterprise information systems have highly unpredictable, non-linear behavior where even minor occurrences might have major implications.

The same phenomena have been observed in the domain of physical and biological systems, be it either an individual animal or a swarm of cooperating bees, and has been extensively researched in the area of Complex Adaptive Systems [1, 6]. 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 [6]. Agents adapt by changing their rules as experience accumulates and can be aggregated into meta-agents whose behavior may be emergent (not determinable by analysis of lower-level agents). A brief review of the CAS characteristics of several successful EAI projects from a business object technology perspective follows. The purpose of this review is to derive design parameters and patterns for successfully integrating enterprise applications.

CAS Characteristics

Enterprise application integration from a Enterprise Component/CAS perspective requires an understanding of Holland's synthesis of CAS concepts, which are outlined in the following paragraphs.

Aggregation (property). There are two important modes of aggregation in CAS systems, objects and components. 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* or Hofstader's analysis of an ant colony [5]. 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).

Tagging (mechanism). This mechanism facilitates the forming of aggregates, from HTML pages to the mechanisms in CORBA or DCOM that allow inter-object communication and facilitate selective mating (for example, 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.

Non-Linearity (property). Non-linear systems exhibit catastrophic and chaotic behaviors. Traffic flow on the Internet is a prominent example of a non-linear system, leading to predictions of the collapse of the network. Other examples encompass brownouts, system loadings, and scalability effects. Lastly, the rate of construction of software itself is a non-linear phenomenon.

Flows (property). This property designates in its simplest form streams between nodes and constitutes the basis for recycling. Workflows and tags that define and condition flows, exemplify this property. Flows typically have a multiplier effect. Money injected into the economy has an effect out of proportion to the amount, similar to email viruses or other message flows on a network. The recycling effect of flows enables the rain forest, an enterprise information ecosystem, or viral adoption of Internet services and applications. Individual pieces evolve, die, are replaced or reused, constantly changing the characteristics of an information system. 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.

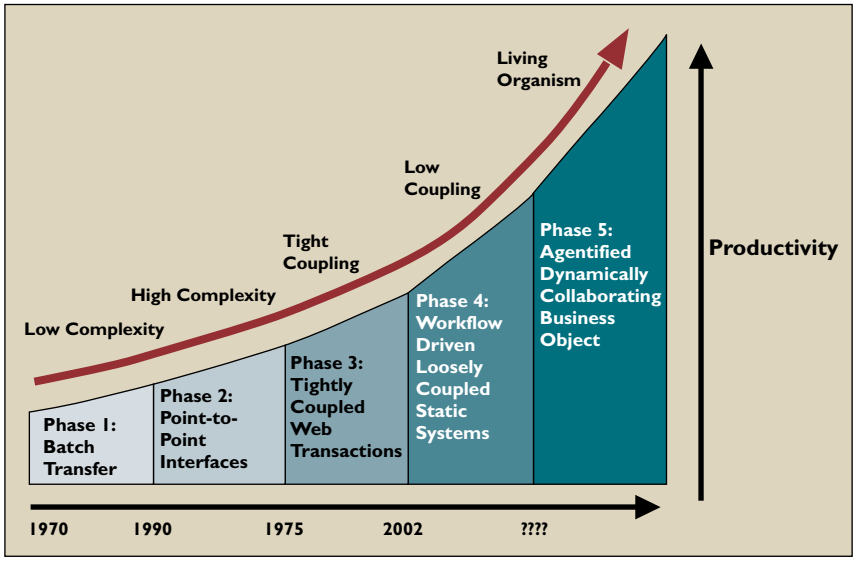

Figure 1. Evolution of EAI.

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 computational 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.

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.

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.

Case Studies

In the following two examples, we reflect on two practical case studies to refine our understanding of CAS in the context of EAI.

Case 1: The Merger of Two Insurance Companies. Van den Enden et al. [6] 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 back-end 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 as the core of their integration strategy. New technology was superimposed on the old in order to enable: interaction between Web HTML clients and mobile WML clients; utilization of standard XSLT transform mechanisms; easy integration with future systems in electronic marketplaces through XML; and 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 back-end system). All details of the front-end or back-end systems are hidden from the workflow layer. Thus Forte Conductor "conducts" the

execution of a process instance by human and machine interactions.

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

In this architecture, two legacy systems are aggregated into a single system by adapters controlled by the workflow engine. Non-linear 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.

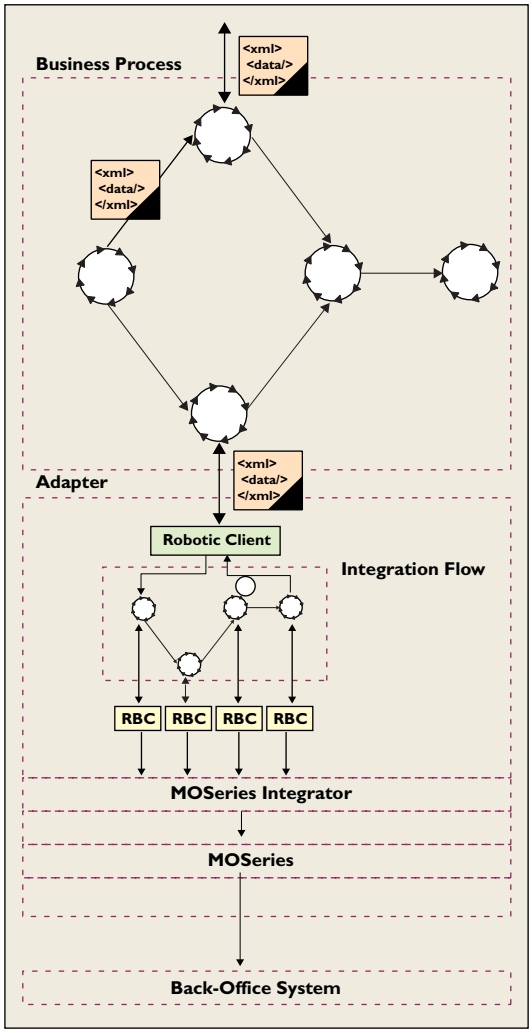

Figure 2. Overview of back-office adapter [10].

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.

Case Study 2: Mobile Device Platform for Integration of Hospital (Legacy) Information Systems. The Internet proliferation 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 impenetrable 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 health care. Over 95% of clinicians have no automation at the point of care in the U.S. versus more than 90% with some level of automation at the point of care in the U.K. and Canada. The net result is a tremendous loss of money due to unbilled or incorrectly coded charges (an average of 8–10% in large integrated health care delivery networks). Loss of funds in failed reimbursement for laboratory tests due to miscoding or failure to demonstrate medical necessity can result in the loss 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 U.S. and that minimal levels of automation would reduce these deaths by 50–80%. Patient safety has been declared a national emergency by the National Academy Institute of Medicine [8]. 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.

Solving this problem requires a complex EAI 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 or PocketPC and the number is growing quickly. Third, the cost of mobile devices and wireless communication is dropping rapidly to the point where total cost of ownership (TCO) is 20% 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 computer professionals could possibly provide in terms of reduction of human pain and suffering. 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 errors. An integration platform architecture for support of mobile/wireless applications at the point of care is shown in Figure 3.

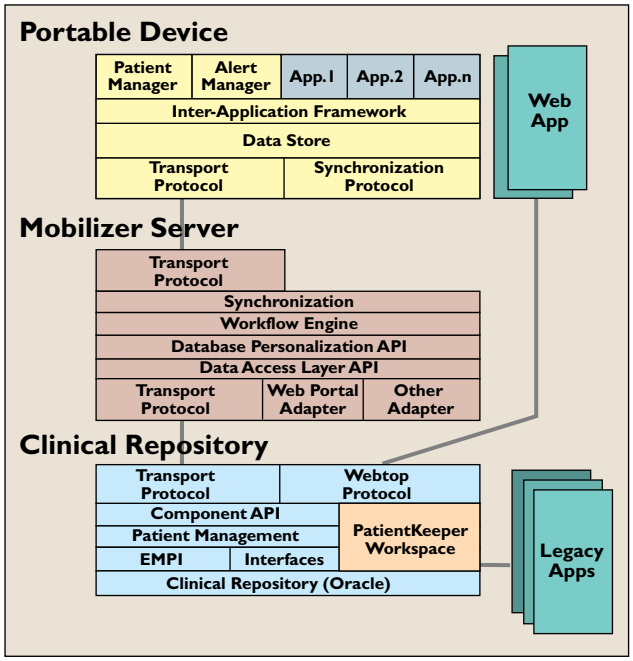

Figure 3. The PatientKeeper architecture.

The top layer of this architecture is the mobile device. While the platform needs to support a wide variety of device types, clinicians only want one device in their coat pocket. The handheld device provides a mobile patient index integrated with back-end legacy systems and an application API, which allows independently authored applications to plug into the framework. All applications are interoperable in the sense that the clinician selects a patient, subsequently selects an application that automatically understands the patient context, and applications share both data and functionality. Key requirements are a single sign-on and security infrastructure, application interoperability, and "always-ready" operations. When the device is disconnected from the network, applications run off the local device datastore. When the device is reconnected by any wireless or wired mode, it automatically synchronizes with back-end 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 toward cardiology. He wants to see his own patients on his Palm Pilot or PocketPC 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 physician's 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 EAI between dozens of independent application systems. 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 the ambulatory clinic, he wants to see data flowing to and from the clinic financial and clinical systems. Clinic systems are typically isolated from hospital systems. Nevertheless, physicians want all this to be transparent to whichever hospital or clinic they are in at the moment.

The clinical repository in this example is integrated using a component object strategy. The repository provides OO component interfaces to the synchronization server. The messaging protocol between the synchronization server and the repository is XML using SOAP as an RPC mechanism.

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 back-end 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 non-linear 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 enterprise computing platforms.

Discussion and Outlook

Currently, we would not expect emergent (non-linear) behavior in today's integrated enterprise systems because of the fact that workflows are generally hardcoded in the integrated applications and internal models are implicitly defined and fixed. Despite this shortcoming, the integrated component based applications did implement some of the remaining CAS properties, as exemplified in the case studies here. These case studies indicated that enterprise components served as a sound basis for future integration.

Comparing these systems with CAS has taught us they require some extensions to support agents with adaptive internal models to dynamically develop and deploy loosely-coupled workflows within the limits of defined adapters. In particular, business object systems need to implement a fundamental concept that will be characteristic of future adaptive systems—automated workflow drives business processes—in place of human interaction. Only in this way, enterprises can achieve the highest level of integration that allows flexible "living" virtual alliances with loosely coupled aggregated and intelligent emergent behavior. Furthermore, the case studies were excellent examples of "capturing" the software of legacy systems and repurposing them for future flexibility. Consequently, legacy investments were preserved while freeing legacy systems from many limitations.

Based on these observations, we strongly believe that static implementations based on tightly coupled enterprise components do not allow systems to adapt operations based on the dynamic state of the runtime environment. Next-generation systems must allow autonomous business object components to decide with whom to collaborate, what services to offer, what services to request, and what visible behaviors to exhibit. To accomplish this, enterprise component assemblies need to be "agentified." Coordination between components is carried out by conversations based on well-known coordination strategies [7]. These strategies encompass composition protocols that prescribe the way in which such agentified business applications interact while allowing dynamic adaptation. Intelligent agents constitute the next higher level of abstraction to component technology, which permit them to negotiate with other agents about their (mutual) goals [9], and even refactor their own goal-oriented behavior.

The main challenge of CAS is to provide the capability to deal with emergent behavior of business systems. Well-designed CAS properties embedded in enterprise systems could bring us closer to a software architecture that could evolve as quickly as business processes are now evolving in 21st century organizations. This could help provide global productivity gains that have been elusive in all but the high tech hardware sectors of our economies. C

References

1. Arthur, W.B. On the evolution of complexity. In *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, Reading, MA, 1994.
2. Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. *Web Services Description Language (WSDL)* 1.1 W3C Note 15 March 2001; www.w3.org/TR/wsdl
3. Eeles, P. and Sims, O. *Building Enterprise Components.* Wiley, New York, 1998.
4. Fayad, M., Hamu, D., and Brugali, D. Enterprise frameworks characteristics, criteria, and challenges. *Commun. ACM 43,* 10 (Oct. 2000), 39–46.
5. Hofstadter, D.R. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, New York, 1979
6. Holland, J.H. *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley, Reading, MA, 1995.
7. Jennings, N., Sycara, K., et al. A roadmap of agent research and development. *Autonomous Agents and Multi- Agent Systems, 1* 1 (Jan. 1998), 7–38.
8. Kohn, L.T., Corrigan, J., et al. *To Err Is Human: Building A Safer Health System.* National Academy Press, Washington, D.C., 2000.
9. Papazoglou, M. Agent-oriented technology in support of e-business: Enabling the development of "intelligent" business agents for adaptive, reusable software. *Commun. ACM 44*, 4 (Apr. 2001), 71–77.
10. Van den Enden, S., Van Hoeymissen, E., et al. A case study in application integration. In *Proceedings of the OOPSLA Business Object and Component Workshop, 15th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, Minneapolis, 2001.

Jeff Sutherland (jeff.sutherland@computer.org) is the Chief Technology Officer for PatientKeeper Inc., Brighton, MA.
Willem-Jan van den Heuvel (wjheuvel@kub.nl) is an assistant professor at the Infolab in the department of Information Systems and Management, Tilburg University, The Netherlands.