Coordinating COTS Applications via a Business Event Layer

Wilfried Lemahieu, Monique Snoeck, Frank Goethals, Manu De Backer, Raf Haesen, and Jacques Vandenbulcke, *Katholieke Universiteit Leuven*

Guido Dedene, *Katholieke Universiteit Leuven and University of Amsterdam*

By introducing an additional abstraction layer in the interaction stack and designing business processes as sequences of events, the BECO approach to COTS integration offers consistency and flexibility.

The shift from custom application development to using COTS products is increasingly commonplace. Unfortunately, whereas we rarely undertake in-house software development without a meticulous analysis and design phase, we often overlook that step when we're planning to integrate COTS applications in an information system. Still, the design aspect remains just as valid: if we integrate COTS software ad hoc, the global information system might be unable to efficiently coordinate its constituent applications and guarantee consistent processing in each of them.

Organizations tend to acquire COTS products on the basis of their functional domains, with each individual application covering a particular domain. Most integration architectures use message-oriented middleware (MOM) to automate the information flow between organizational units via message exchanges between COTS applications. As such, the message exchanges become the integration architecture's units of coordination. This article explains how the abstraction level of these one-to-one message exchanges is too low to efficiently design an integration architecture. As an example, the creation of a purchase order affects order management, accounting, marketing, and many other functional domains. If all corresponding COTS applications are to process the order in a coordinated way, designing the interaction requires several one-to-one message exchanges and becomes quite intricate.

Therefore, we propose BECO (*b*usiness *e*vent-based *co*ordination), an approach to integration architecture design that is based on the concept of *business events*. These act as higher-level units of coordination that enforce consistent processing in all participating applications. Underneath, it reuses existing one-to-one communication technologies for event notification. We'll also show how to design and represent *business processes* concisely—as sequence constraints on business events. The result is a consistent, flexible integration approach that you can layer entirely on top of existing technologies.

A real-life case

A particular telecommunications company

0740-7459/05/$20.00 © 2005 IEEE

positions itself as a broadband application provider for the small- and medium-enterprise (SME) market.[1] It's organized around four key business units: Sales and Marketing, Service Provisioning, Finance, and Customer Services. Figure 1 depicts the main business process and the resulting information flow. The Sales and Marketing business unit sets prices, completes sales transactions, and notifies the Service Provisioning and Finance business units of ordered products. The Service Provisioning business unit coordinates the installation of all telecommunication services ordered. It notifies the Finance business unit of completed installations and the Customer Services business unit of the installed configurations. Finance handles invoicing, and Customer Services is responsible for after-sales service. Apart from Sales and Marketing, which only uses elementary office software, each business unit relies on COTS software from different providers.

Although each software package can support its particular business unit well, the lack of integration and coordination between the different standalone applications is problematic. For example, consider the storage of people data. Sales and Marketing needs to store data about whom to contact for commercial sales at a customer company. The Service Provisioning and Customer Services applications maintain data on technical contacts in that company. The Finance application keeps track of financial contacts. Because the company mainly deals with SMEs, one individual might fill several of these roles, so the data about that person will be scattered across several business units. The company could use an approach ensuring consistent, company-wide data manipulation. This affects not only the data but also the behavioral and coordination aspects of the company's global information system. For example, the company might want to ensure that people who misbehave in one of their roles won't be allowed to take on a, say, financial contact role in the future. Such a policy requires integrated, coordinated processing of "blacklisting" actions among all COTS applications.

Integrating enterprise applications

Typically, you would integrate COTS applications via enterprise application integration technologies such as *remote procedure calls* or MOMs. RPCs represent interactions as procedure calls from one component to another (remote) component. Its most recent incarnations are distributed-object technologies such as remote method invocation (RMI).[2] With MOM, applications interact by exchanging messages. Recently, many MOM products have evolved into integration brokers. A business process specification document captures the specification of which applications are to exchange which messages and in what sequence. The integration broker enacts the specified business process by managing the desired sequence of message exchanges. Web Services apply a similar approach. These interact through SOAP,[3] which is based on exchanging XML messages over the Internet. As application designer, you can specify the business process in an orchestration language such as BPEL4WS (Business Process Execution Language for Web Services).[4]

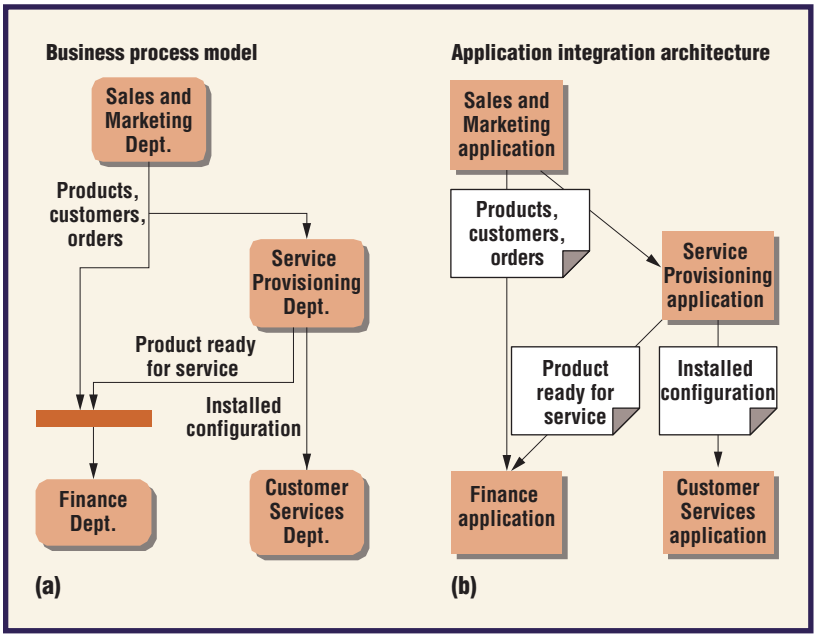


Figure 1. Information flow in a telecommunications company: (a) the main business process and (b) the application integration architecture.

All the technologies we've mentioned essentially represent one-to-one communication paradigms. In each basic interaction, there's one sender (or procedure caller) and one receiver (or procedure callee). The individual message exchanges (or procedure calls) are the units of coordination: an integration broker executes, manages, and monitors the interaction between the COTS applications at the level of these message exchanges. They are also the basic building blocks of the process definition: a business process is defined as feasible sequences of message exchanges.

However, defining the business process at the individual message exchange level entan-

BECO defines a business event as a real-world phenomenon that requires coordinated processing in one or more applications or components.

gles the process's respective business activities with the components' invocation mechanisms. Moreover, the abstraction level of the message exchanges doesn't allow easily for the design of activities that involve coordinated processing in multiple applications. For example, blacklisting a person might affect people data in all business units, so all corresponding COTS applications should be notified. Also, any business unit could initiate a request to blacklist someone. So, blacklisting can be considered a many-to-many interaction, which might involve different sequences of message exchanges depending on which business unit takes the initiative for the blacklisting and how the message is propagated from application to application. However, with these technologies, which only coordinate interaction at the level of individual message exchanges, it's impossible to abstract the blacklisting "event" itself from the various possible notification patterns.

Another example is the creation of a purchase order. This would require some processing in each COTS application. Moreover, each business unit may impose its own business rules and constraints on creating an order. For example, the Finance application may refuse orders from customers whose unpaid outstanding orders exceed a certain limit, the Service Provisioning application may refuse technically unfeasible orders, and the Sales and Marketing application may reject orders with inappropriate price setting.

So, creating an order requires application coordination that surpasses what one-to-one messaging can achieve. Moreover, you wouldn't want the Service Provisioning application to accept and further process an order after the Finance application refused it. Either all applications should agree with creating the order or none should. While in theory you could achieve this "all or nothing" coordination by means of a multitude of one-to-one messages, it would be intricate to design and debug, let alone verify conceptually. This situation requires a higher-level unit of coordination representing many-to-many interactions.

Note that a many-to-many pattern exists in so-called publish-subscribe MOM products. However, their inherent fire-and-forget approach makes them less suitable for situations that require a coordination aspect. For example, the middleware could dispatch a `blacklist_customer` or `create_order` message to all applications involved (the many-to-many aspect), but the coordination doesn't stretch beyond guaranteed message delivery. These products don't coordinate message processing in the respective applications. Our integration approach combines many-to-many interaction with the ability to coordinate processing and enforce consistency over the participating applications.

Business events

BECO defines a business event as a real-world phenomenon that requires coordinated processing in one or more applications or components. Business events are atomic in the sense that they should never be processed just partially. All parties involved in a business event may enforce business rules and constraints as preconditions on the event. If the event satisfies all preconditions, all participants process it. If it violates one or more preconditions (for example, if the Service Provisioning application views the ordered product combination as technically unfeasible), the entire system rejects the event, and no processing takes place in any of the applications. (If processing has already occurred, it's rolled back). In this sense, events incorporate a transactional aspect.

For example, consider the atomic business event `create_order`. Although the system notifies several applications of the event (Finance verifies the limit on outstanding orders, Service Provisioning checks technical feasibility, Sales and Marketing monitors price setting, and so on), the event is either accepted or rejected in its entirety. The overall system would revert into an inconsistent state if, for example, the Service Provisioning application successfully processed an order that the Finance application rejected.

Each business event represents a many-to-many interaction that can be decomposed into a sequence of one-to-one interactions to notify participants and verify preconditions. However, you can abstract this notification aspect for someone interested only in the global activity. A customer, for example, might want to know only whether his or her order has been accepted or rejected, not which message exchanges took place underneath.

We can easily design the event-based integration of components using a *component-event table*. The CET's columns identify the

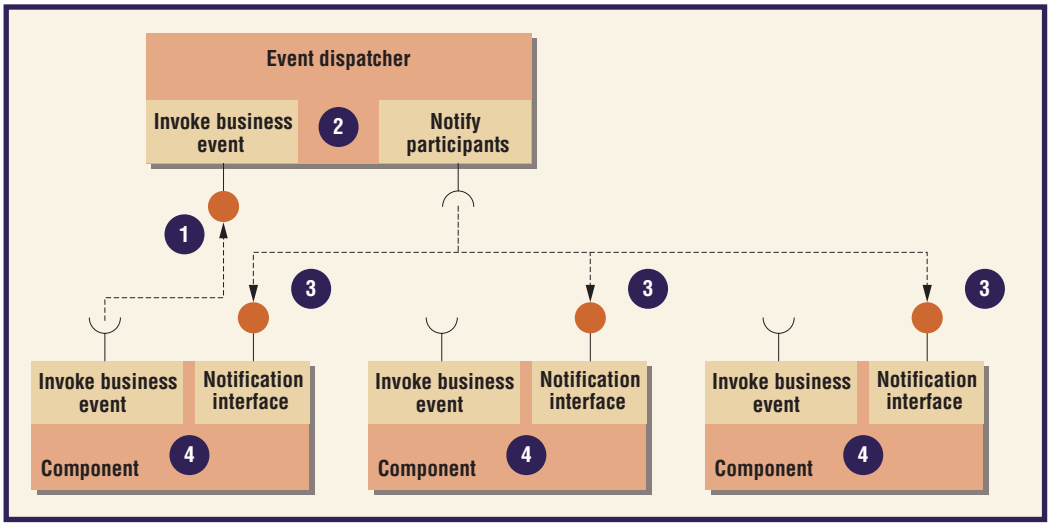


Figure 2. General event-dispatching execution scheme.

components or applications that we must integrate, and the rows identify the business events. The cells denote which application is involved in realizing which event. Table 1 presents a (simplified) CET for the sample case.

In this way, we introduce an additional abstraction layer in the interaction stack. The *business event layer* deals with business events—that is, coordinated many-to-many interactions. The *notification layer* deals with one-to-one message exchanges (or RPCs), which the system uses to notify the appropriate applications of a business event. Possible event parameters (for example, `create_order` might have `product-id`, `customer-id`, and `price` parameters) become attributes to the messages or procedure calls that embody the notifications. The CET defines the mapping between abstraction layers: each business event gives rise to several message exchanges, one for each marked cell in the corresponding row in the CET.

An *event dispatcher* notifies participating applications by initiating the appropriate message exchanges and coordinates their business event processing. Figure 2 represents the general execution scheme:

1. A component triggers a business event by sending a message to (or invoking a procedure on) the event dispatcher.
2. The event dispatcher verifies whether the event satisfies the preconditions imposed by all participating components (as the CET denotes).
3. If so, it notifies all participating components of the event by sending a message to (or invoking a procedure on) each of them.
4. Upon notification by the event dispatcher, each component processes the business event internally.

If one or more preconditions aren't met or if one or more components fails to process the event, the event dispatcher organizes a global rollback or some compensating action, so that no trace of the failed event remains in any component. In

Table 1

A component-event table for the sample case.

Component or application	Sales and Marketing	Service Provisioning	Customer Support	Finance
`create_order`	✔	✔		✔
`cancel_order`	✔	✔	✔	
`install_completed`		✔	✔	✔
`invoice`			✔	
`pay`			✔	
`Create_customer`	✔	✔	✔	✔

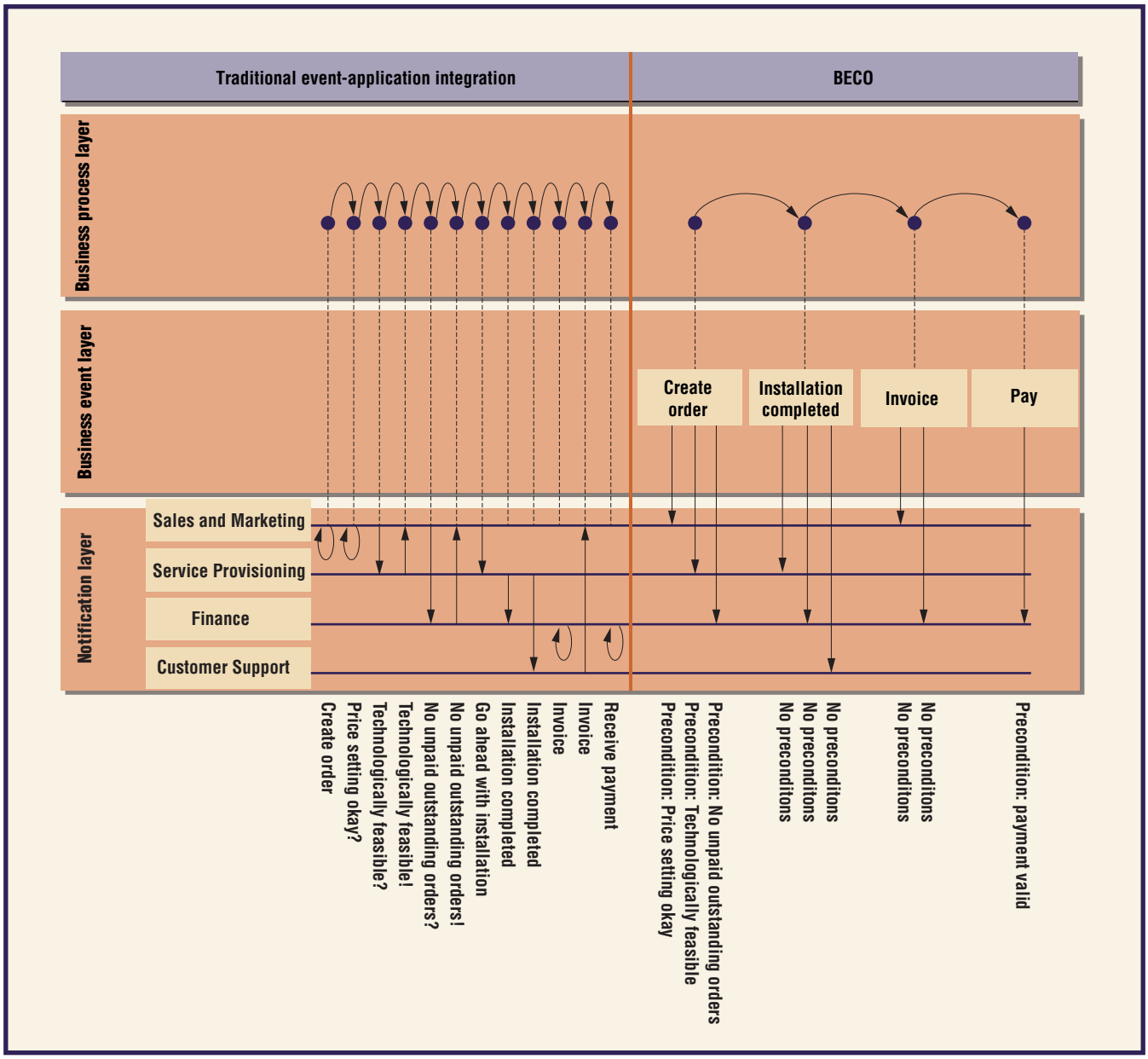


Figure 3. Layers in traditional enterprise application integration versus layers in BECO.

this way, events shape a transactional coordination mechanism, derived from the business model, over the individual notifications.

Business processes

Identifying business events as coordination units yields a concise paradigm for representing (and enacting) business processes. Indeed, sequence constraints, a particular kind of precondition, determine the order in which business events may occur. Thus, we can represent a *business process* as feasible sequences of business events. For example, the system will refuse an `installation_completed` event if a `create_order` event with the same `order-id` doesn't precede it. Likewise, an `invoice` event should precede a `pay` event. We can use various formalisms—for example, finite-state machines, regular expressions, or Petri nets—to express sequence constraints on business events.[5]

Regardless of the representation format we choose, what's important is that the sequence constraints apply to business events. In this way, we can separate the underlying notification aspect from the business process sequencing. Traditional orchestration languages such as BPEL4WS define business processes by enforcing

sequence constraints on individual message exchanges. However, any many-to-many interaction between *n* applications will yield more than *n*! possible one-to-one message exchange patterns to establish communication. So, the process descriptions are needlessly complicated because they are intertwined with the individual business events' notification schemes. BECO decouples event sequencing (the business process) from notification sequencing (the messaging pattern that notifies the respective applications that participate in one particular event). Figure 3 illustrates this difference.

The result is a three-layered approach to application integration. The notification layer exploits existing one-to-one MOM or RPC technologies for event notification. The layer above is the business event layer, which considers a business event a coordinated many-to-many interaction. The event dispatcher establishes the coordination and mapping between business events and notifications by consulting the CET. Finally, the business process layer defines the business process, specified as sequence constraints on business events. The event dispatcher coordinates compliance with the sequence constraints.

This layered architecture and separation of concerns yield a highly flexible environment in which business process, event participation, and notification patterns can all change independently. That is, you can redesign a business process by manipulating sequence constraints, changes in a single activity's execution will affect the way a business event is implemented by means of event notifications, and changes in the participating applications will result in modified CET cells.

Implementation aspects

Our sample case, discussed previously, has a fully operational, J2EE-based implementation. We conceived the event dispatcher as a set of session Beans that interact with the COTS applications through an RMI-based notification layer. For each COTS application, we wrote a coordination agent that interfaces the notification layer with a particular COTS component. The coordination agent acts as an intelligent adapter, supporting three types of interaction.

First, the agent translates event notifications into calls to the COTS application's native API. This was quite straightforward to implement. Having a *reactive* COTS application suffices—that is, one that offers an API for responding to external requests.[6]

Next, the coordination agent "listens" for internal actions in the COTS application (induced, for example, by the application's own user interface) that are relevant for the rest of the system—actions from which the system should infer a global business event. This more complex action requires a proactive COTS application[6]: it should be able to initiate requests to other components to provoke a business event. Fortunately, the COTS applications we used in our sample case either had this ability or we could create triggers in their underlying database to provide this functionality. However, in general, many COTS applications are reactive and wait for service requests. This doesn't necessarily invalidate BECO's applicability: advanced wrapping techniques can turn reactive COTS applications into proactive ones.[6]

Finally, the coordination agents can support query functionality for the COTS components' internal data. Part of these data is also replicated in a "central" relational database, accessible through GUI components developed in-house. End users can induce several business events directly from these GUI components, which in turn interact with the session Beans. (For a detailed discussion of implementation, see elsewhere.[1])

B2B event-based coordination

So far, we've discussed BECO as an (intra-) enterprise application integration technology. However, we can extend the approach to handle (interenterprise) business-to-business integration (B2Bi). We briefly summarize the most important issues.

BECO is fully compatible with Web Services technology, which is becoming the de facto standard for B2Bi. In this context, we can implement the event notification layer through SOAP messaging. In a static form of B2Bi, all interacting partners "know" one another in advance. The CET is drawn up at deployment time and remains fairly stable. In a dynamic B2Bi situation, partners dynamically find each another, after which they participate in short-lived, ad hoc transactions. In that case, the event dispatcher should contain a subscription mechanism based on dynamic updating of the CET. When a partner's Web service subscribes to an event, the event dispatcher marks the corresponding CET cell; when a service unsub-

BECO is fully compatible with Web Services technology, which is becoming the de facto standard for B2Bi.

About the Authors

Wilfried Lemahieu is an associate professor in the Management Information Systems Group of the Department of Applied Economic Sciences at Katholieke Universiteit Leuven. His research interests include business process management and Web Services, postrelational database systems, and hypertext systems. He received his PhD in applied economic sciences from K.U. Leuven. Contact him at K.U. Leuven, Dept. of Applied Economic Sciences, Naamsestraat 69, 3000 Leuven, Belgium; wilfried.lemahieu@econ.kuleuven.be.

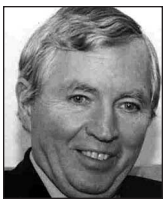
Monique Snoeck is full professor in the Management Information Systems Group of the Department of Applied Economic Sciences at K.U. Leuven. Her research focuses on object-oriented conceptual modeling, software architecture, and software quality. She received her PhD in computer science from K.U. Leuven. Contact her at monique.snoeck@econ.kuleuven.be.

Frank Goethals is a PhD student in the Management Information Systems Group of the Department of Applied Economic Sciences at K.U. Leuven. He's working on extended enterprise infrastructures and coordination issues. His research is financed by SAP Belgium. He received his master's degree in economics from K.U. Leuven. Contact him at frank.goethals@econ.kuleuven.be.

Manu De Backer is a PhD student in the Management Information Systems Group of the Department of Applied Economic Sciences at K.U. Leuven. His research interests are in the formalization and verification of business process modeling and Web Service compositions. Contact him at manu.debacker@econ.kuleuven.be.

Raf Haesen is a PhD student at the KBC-Vlekho-K.U. Leuven Research Center. His research interests are object-oriented conceptual modeling, service and component-based architecture, and model-driven development. He received his civil engineering degree from the Computer Science Dept. at K.U. Leuven. Contact him at raf.haesen@econ.kuleuven.be.

Guido Dedene is a full professor in the Management Information Systems Group of the Department of Applied Economic Sciences at K.U. Leuven and holds the Development of Information and Communication Systems Chair at the University of Amsterdam. His teaching and research focus on consistent formal development of business-oriented information systems as well as the management aspects of such systems. He received his PhD in general relativity theory from K.U. Leuven. Contact him at guido.dedene@econ.kuleuven.be.



Jacques Vandenbulcke is a full professor in the Management Information Systems Group of the Department of Applied Economic Sciences at K.U. Leuven. He also coordinates the Leuven Institute for Research on Information Systems. His research interests include database management, data modeling, and business information systems. He received his PhD in applied economic sciences from K.U. Leuven. He's president of Studiecentrum voor Automatische Informatieverwerking (SAI), the largest society for computer professionals in Belgium. Contact him at jacques.vandenbulcke@econ.kuleuven.be.

scribes, it removes the cell mark. We propose a prototype architecture for an event-based Web Services environment elsewhere.[7]

Dynamic B2Bi requires automated facilities for discovering Web services that are qualified to interact with one's own applications. The services should be compatible, not only in terms of interfacing but also in terms of process. BECO facilitates automated, formal verification of process compatibility by analyzing the sequence constraints imposed by the interacting components. As a trivial example, a Supplier service may not accept a shipping event unless a payment event has taken place. On the other hand, if Customer Service requires shipping to precede payment, these services can't collaborate meaningfully. Elsewhere, we outline a formal verification mechanism based on process algebra.[5]

Although event-based architectures have existed for quite some time,[8–10] none of them use events for business process definition and application coordination in the way BECO does. We've validated the BECO approach in several other projects. Initially, it was conceived in the related discipline of integrating legacy applications with new components. At present, we are applying the same principles in a requirements-engineering and architectural-design project (including support for integrating several COTS applications) for one of the largest Belgian banks. The implementation modalities vary slightly, depending on the targeted environment.

For example, we've experimented with hard-coding the CET into the event dispatcher, but implementing it as a lookup table makes it more flexible. Another degree of freedom is the way in which to verify preconditions and coordinate event execution. Precondition verification can be done centrally by the event dispatcher or delegated to the respective components. The first option is possible only if the event dispatcher can access the respective components' business rules that determine the preconditions. The second option requires a more complex notification pattern, because the respective components must communicate success or failure for their part of the precondition validation. However, the second approach allows for dealing with embedded business rules that are internal to and inextricable from the respective compo-

nents, as is often the case with COTS applications. Regardless of whether preconditions are verified centrally, locally, or in a mixture of both, the event dispatcher coordinates the event's global success or failure and guarantees a consistent commit or rollback in the respective applications. Again, the actual implementation might vary. With centralized precondition checking, the event dispatcher need only dispatch notifications for events that effectively satisfy all preconditions. For decentralized precondition checking, a transactional approach is more appropriate.[5] Again, each of these approaches can be layered over existing middleware technologies.

References

1. W. Lemahieu, M. Snoeck, and C. Michiels, "Integration of Third-Party Applications and Web-Clients by Means of an Enterprise Layer," *Annals of Cases on Information Technology*, vol. 5, 2003, pp. 213–233.
2. E. Pitt and K. McNiff, *Java.rmi: The Remote Method Invocation Guide*, Addison-Wesley, 2001.
3. S. Seely and K. Sharkey, *SOAP: Cross Platform Web Services Development Using XML*, Prentice Hall, 2001.
4. S. Weerawana and F. Curbera, *Business Process with BPEL4WS*, white paper, IBM, 2002.
5. M. Snoeck et al., "Events as Atomic Contracts for Application Integration," *Data and Knowledge Eng.*, vol. 51, no. 1, 2004, pp. 81–107.
6. A. Egyed and B. Balzer, "Integrating COTS Software into Systems through Instrumentation and Reasoning," to be published in *J. Automated Software Eng.*
7. W. Lemahieu et al., "Event Based Web Service Description and Coordination," *2nd Int'l Workshop Web Services, E-Business, and the Semantic Web* (WES 2003), LNCS 3095, C. Bussler et al., eds., Springer-Verlag, 2004, pp. 120–133.
8. A. Carzaniga et al., "Issues in Supporting Event-Based Architectural Styles," *Proc. 3rd Int'l Software Architecture Workshop*, ACM Press, 1998, pp. 17–20.
9. G. Cugola, E. Di Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS," *IEEE Trans. Software Eng.*, vol. 27, no. 9, 2001, pp. 827–850.
10. R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," *Proc. Int'l Workshop Distributed Event-Based Systems* (ICDCS/DEBS 02), IEEE CS Press, 2002, pp. 585–588; www.dsg.cs.tcd.ie/~meierr/publ/docs/meierr_taxonomy.pdf.