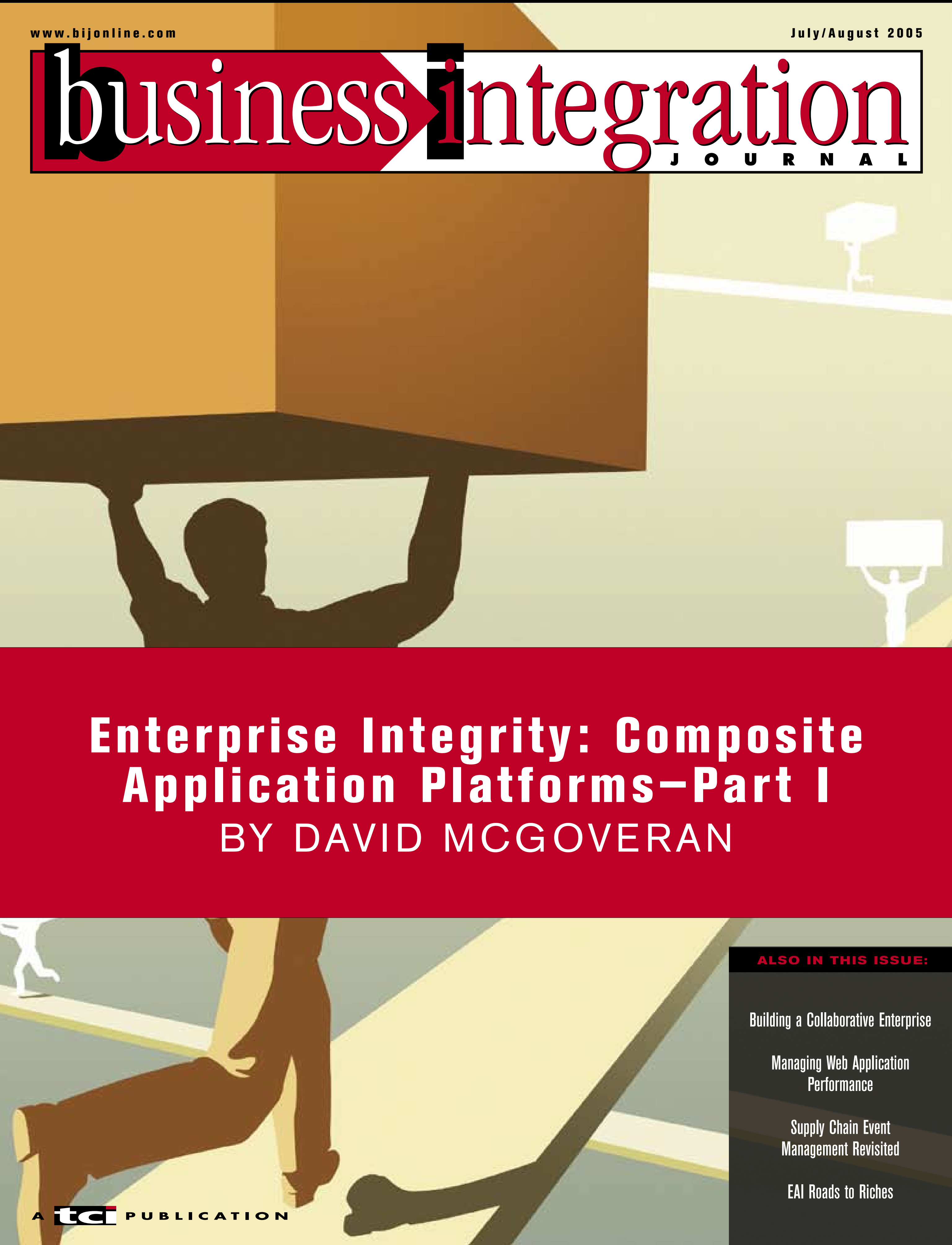
Optimizing Business Integration & Effectiveness Through Leading-Edge Technologies

www.bijonline.com

July/August 2005

business integration JOURNAL



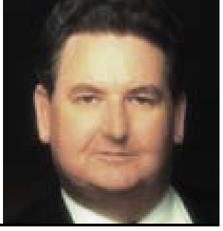
Enterprise Integrity: Composite Application Platforms—Part I

BY DAVID MCGOVERAN

A tci PUBLICATION

ENTERPRISE INTEGRITY

BY DAVID McGOVERAN



Composite Application Platforms: Part I

Numerous articles on composite applications and their benefits have appeared in the pages of *Business Integration Journal*. A Composite Application Platform (CAP) combines the facilities of an integration server and an application server to enable composition and orchestration of services into new applications. It uses a model-driven architecture to develop and orchestrate services. It permits new applications to be built with as little programming as possible, while enabling maximum reuse of IT assets. Composite applications have become one of the primary reasons to deploy a Service-Oriented Architecture (SOA). While an SOA is the preferred implementation of a CAP, it's certainly possible to use other component-based architectures. Over the next few issues, we will treat composite applications as being composed of services and, therefore, implemented using an SOA, and describe the components you should be looking for in an ideal CAP.

The primary benefit of a CAP is to enable reuse of IT software and data assets to enable rapid design, redesign, and deployment of new business functionality. The result is lower cost of IT operations and improved IT responsiveness. This strategy demands a uniform conceptual abstraction above the technical detail of those assets so that services can be composed and orchestrated without low-level programming. The environment must provide context management and coordination across the composite application's services, any number of which may comprise a unit of work. Composite applications blur the distinction between application development and application integration, and so combine the facilities and methodologies of each.

A CAP provides a robust environment for the rapid design and use of composite applications. It comprises design tools, methodologies, services and processes, an abstraction layer that presents components and services in a uniform manner, and various libraries. It's possible for a CAP to form the IDE and integration layers of a BPMS, with the CAP orchestration modeler and orchestration engine being special, restricted cases of the BPMS process modeler and process engine. However, don't confuse the two. Despite the frequent misuse of "business process" by those vendors and standards bodies, the business and IT concepts of process deserve careful differentiation. As an IT tool, a CAP orchestrates IT processes that rarely have the behavioral complexity of business processes.

Taken together, this definition of CAP implies that, in order to be considered in the CAP category, a product should have certain design time, run-time, management, and support components. It will take several columns to discuss the necessary components. This month, we'll begin our discussion of the design time components, completing it next month. We'll highlight the discussion of each component with a bullet so they'll be easy to locate throughout the series of columns.

The primary component of a *CAP design time environment* is the orchestration modeler:

- **Orchestration Modeler:** The orchestration modeler is used to capture, design, and modify service properties and the orchestration of those services through a rich graphical depiction of services and controlled flows among them. The tool's modeling constructs should reflect the development paradigm so that the orchestration can be implemented directly from the model. The tool should promote and actively support at least one design methodology, but shouldn't restrict the organization from using its preferred methodology. It should permit users to define and selectively enforce relevant design standards such as BPMN and UML. Process abstraction via nested processes should be possible so an orchestration can be created and maintained in successively more detail, without forcing the user to grasp all of it at once. Process independence should allow the designer to separate deployment-specific considerations from application requirements. The design platform should be integrated with the run-time environment so the application, or portions of it, can be safely tested and deployed. The model should be used to drive run-time orchestration without requiring an intermediate compilation or application redeployment. Similarly, if generation of the composite application from the model is a development or deployment option, its deployment should not affect the uptime of other applications. Otherwise, the agile, incremental change and high availability that a CAP promises is compromised.

Next month, we'll discuss the remaining design time components, including transformation and transparent data access modeling, transaction modeler, portal designer, and integrated development environment. CAP design time components should be seamlessly integrated so the user can move between them without losing context and without having to use a different user interface style.

Until next column, may your *enterprises* be orchestrated and your *integrity* unyielding. **bij**

About the Author

David McGoveran is president of Alternative Technologies. He has more than 25 years of experience with mission-critical applications and has authored numerous technical articles on application integration.
e-Mail: mcgoveran@bijonline.com; Website: www.alternativetech.com