Optimizing Business Integration & Effectiveness Through Leading-Edge Technologies

www.bijonline.com

October 2005

business integration JOURNAL

Enterprise Integrity: Composite Application Platforms—Part III

BY DAVID McGOVERAN

ALSO IN THIS ISSUE:

10 Best Practices for Creating an SOA

The Power of Information in Supply Chain Collaboration

The Return of B2B: How Four Industries Rate

A tci PUBLICATION

ENTERPRISE INTEGRITY

Composite Application Platforms: Part III

BY DAVID McGOVERAN

In the previous two columns of this series, we've examined the design time facilities required for a Composite Application Platform (CAP). A CAP is an essential component of an SOA, one that enables the service definition, reuse, orchestration, and maintenance required for an agile IT infrastructure. This month, we turn our attention to the run-time facilities, completing the task next month. Of course, the design time and run-time facilities need to be seamlessly integrated so the developer can create, test, deploy, and maintain composite applications without resorting to costly and risky changes from development to production environments.

In a well-integrated, model-driven design, development, and deployment environment, run-time facilities can be conceptual rather than physical. These facilities are hardly distinguishable from those used for system test and debug; the differences being more a matter of preferences and configuration than capabilities. Realizing the advantages of SOA requires an iterative, incremental approach to service composition in which a running system can be extended or modified component by component without risk to the overall application. Failing this, SOA becomes little more than a conceptual architecture replete with numerous complexly interacting standards.

A CAP should include the following run-time facilities:

- **Orchestration:** The services that comprise a composite application need to be managed at run-time according to an orchestration model, implementing the composite application by managing context, the real-time invocation (or activation) and termination (or completion) of the component services, and service interconnection (including data access and transformation). All aspects of run-time orchestration should be model-driven. Orchestration shouldn't limit the methods by which services are implemented, the manner in which they are invoked, or the messaging protocols used. Orchestration includes collaborative context management, enabling services to transfer or share a context at run-time, thus providing appropriate cohesion while maintaining the loose coupling and late binding essential for composite applications. The execution language can limit the ability to compose services that run the way the business expects, creating instead a functional but unrecognizable IT parody of business processes.
- **Interface Management:** A CAP is of little value if the orchestration engine can't communicate with services as implemented. It must be able to communicate both control flow and data flow in a coordinated fashion, though these may be separately defined and quite distinct. Support for a variety of transports and end-point adapters (e.g., messaging, middleware, files and databases, applications, and presentation software) is essential in today's IT and is an interface management issue.
- **Service Access:** Underlying systems should be decoupled from business applications, components, and processes, permitting composite applications to integrate with any existing infrastructure. A central interface controls communication with participating systems via an extensible exchange infrastructure such as that provided by Web Services. Composite application designers shouldn't need technical knowledge of component or service implementation, or interfaces, whether native or external.
- **Monitoring:** The ability to detect and capture data relating to key events within the composite application is highly desirable in a CAP, enabling both technical and business managers to more effectively manage the use of resources. More often than not, such events will require some form of analytic computation in order to convert them from raw measurements into business metrics.
- **Dashboards:** A facility to design and display the status of monitored orchestration instances and the metrics they produce is needed not only by business managers, but by technical and system administrators as well. It should be possible to generate printable reports as well. Portal-based dashboards are a common implementation.
- **Native Run-Time Environment:** A distributed, highly available enterprise-class application server or enterprise service bus is the primary run-time environment for a CAP, including the orchestration, interface management, and native components. The native run-time environment should support instance pooling, load balancing across multiple instances, standard interfaces, and messaging protocols.
- **Repository:** A CAP requires a sophisticated DBMS and meta-schema repository to manage and enable appropriate reuse. Many data objects must be stored by the repository, including orchestration definitions, integrity rules, instance histories, messages and data flows, business metric definitions and data, business analytic and dashboard definitions, along with saved data, transaction definitions and data, security and policy definitions, access histories, error events and resolutions, and so on.

Next month, we'll look at more run-time facilities and some additional concerns. Until then, remember: Well-designed CAPs accelerate SOA adoption and financial return for your *enterprise*, but only if the interfaces are elegant and the internals have architectural *integrity*. **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