Optimizing Business Integration & Effectiveness Through Leading-Edge Technologies

www.bijonline.com

September 2005

business integration JOURNAL

Enterprise Integrity: Composite Application Platforms—Part II

BY DAVID McGOVERAN

ALSO IN THIS ISSUE:

SPECIAL FORRESTER RESEARCH SECTION

EII vs. ETL: No Contest

Creating Standards-Based SOA Governance

10 Factors to Consider When Comparing Enterprise Portal Servers

A tci PUBLICATION

ENTERPRISE INTEGRITY

BY DAVID McGOVERAN

Composite Application Platforms: Part II

Last month's column began a series on Composite Application Platforms (CAPs). We discussed a CAP's purpose and benefits, and began a detailed look at the design time components with the most important, the Orchestration Modeler. The top-level user interface, it is the primary determinant of a CAP's ease of use and productivity. To be most effective, it should provide conceptual abstraction and integration. Additional services (if well-integrated) determine its functional breadth. This month we discuss the remaining design components: transformation and transparent data access modeling, transaction modeler, Integrated Development Environment (IDE), and portal designer.

- **Transformation and Transparent Data Access Services:** A composite application often draws data in incompatible formats from disparate data stores. Data passed between services by the CAP may need to be transformed. Transformations can be exceedingly complex, requiring support for multiple sources and targets, semantic rationalization, staging and synchronization, etc. An intuitive design interface should be provided with automatic error checking. Transformations should be represented as services invocable within an orchestration. Since transformations are an IT artifact, it should be possible to defer their definition, and even hide them in the orchestration's representation. Data access and transformation is preferably driven by live metadata, rather than relying on developer-entered data descriptions. Rule-based and parameter-driven data transformation services are essential in all but the simplest of composite applications. The CAP should support an extensible library of transformations. Access to an Enterprise Information Integration (EII) can mitigate some of the need for these services.
- **Transaction Definition and Management:** Real business applications require enforcement of units of recovery (physical transactions), units of consistency (logical transactions), and units of audit (business transactions). Arbitrary units of work composable across diverse components and services, and under the control of multiple resource managers or business entities, should be possible. In many environments, both tightly coupled distributed transactions and long-running transactions must be supported. Multiple transaction models (tightly coupled distributed transactions, compensation spheres, and collaborative transactions) and environments (such as CORBA, CICS, WebLogic/Tuxedo, Enterprise JavaBeans [EJBs], or .NET) should be supported. Developers should be guided to use an appropriate transaction model based on declarative specification of requirements, rather than forced to understand the intricacies of and interactions among these technologies. Recovery methods should be predictable and guarantee consistency across orchestrated components and services. My "Business Transaction" series (March-September 2004) provides more information.
- **IDE:** A model-driven design and development tool suite is needed to develop new services or components and to prepare existing services for discovery and interoperation with the orchestration engine. The IDE should hide complexity (e.g., the details of EJBs, Web Services Definition Language [WSDL], SOAP, etc.), preferably through a combination of a model-driven methodology, a high-level conceptual model, wizards, a library of patterns (orchestration, service, and transformation), and default templates. In simplest form, such an IDE enables new services that are *orchestration-aware*. With more sophistication, an IDE for designing and developing services that are event-driven and rule-based applications or application components is highly desirable. An intuitive user interface should hide the complexities of diverse components from most users through a single development abstraction. Developers should be able to rapidly encapsulate existing software assets, deploying them as if they were native. These features accelerate composite application design, deployment, and execution through asset reusability (regardless of original deployment technology) and automatic configuration.
- **Portal Designer:** A common facility is required that implements end-user interactions as composable services. A portal integration tool with the user interface logic implemented as Web Services is standard. Additionally, portal support offers some form factor independence: Mobile client support may require no more than redeployment for the particular interaction device, but requires run-time support for data synchronization (e.g., bi-directional replication) and asynchronous connections.

An effective CAP won't limit services integration to support for a single services interaction model or make assumptions about service complexity. Orchestration standards are immature and unrealistic, failing to adequately support human interaction and workflow, complex business processes, transaction management, error and exception handling, etc. A CAP that merely implements some orchestration standard jeopardizes the *integrity* of real *enterprise* applications, and ultimately will fail to deliver on its promises. Next month, we'll examine some run-time requirements. **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