Alternative Technologies

Enterprise Integrity: Architected Scalabilty

Vol. 2, No. 7

Last month's "quick evaluation guide" to integration architectures had little space to discuss the myriad possible architectural choices that face an EAI vendor and their customers, let alone all the implications of those choices. Nonetheless, a few general statements can (and should) be made regarding the impact of architectural choices on scalability and performance. In order to keep this coherent, I'm going to follow the same presentation format and software stack that I followed last month. You might want that issue at hand as you read. Hold your seats – this is going to be very opinionated!

Architecture – If the functional architecture cannot be described in a block diagram, using hierarchies to organize and simplify, you are going to have lots of scalability and performance (not to mention maintenance and ease of use) problems. Additionally, your system is not going to scale functionally if it's architecture isn't uniform: that means integration will eventually become a costly headache. A peer-to-peer, service oriented architecture meets these needs best, permitting minimization of costly message passing and minimization of message overhead.

Integrated Design/Development/Deployment Environment – If you might think the IDE has little to do with scalability and performance, you're wrong. When user written extensions (a.k.a. integrations) and product extensions are developed with the same tools, that great bane of scalability and performance – known as impedance mismatch – can be minimized.

System support facilities – A great way to add overhead to a system is to develop functionally redundant and variant support facilities, almost a certain outcome if standard system support facilities are not supplied. Such redundancy adds unnecessary code that must be supported and increases the likelihood of context switches. It almost certainly will decrease API uniformity, an essential element of good scalability and high performance. With every additional interface conversion or wrapper, we increase the execution path length for business functions. Call it slowww…

Intelligent adapters (a.k.a. connectors) – Good adapters can do a great deal to minimize message volume and message overhead. If the adapter is an intelligent agent, it can be interrogated and remotely, dynamically configured to work the most appropriate way. A more or less autonomous adapter enables availability and higher performance. Tasks like message reformatting and data transformation might be done locally, with the output routed directly to the next business application (versus a hub).

Transport – Not every method of communicating messages is appropriate for every instance of application-to-application communications. While asynchronous messaging (and in particular publish/subscribe) is very flexible, it is not very helpful when transactional integrity

requires synchronization among distributed business functions. Flexible, configurable choice in the transport can make the difference between timely and slow, unreliable integration.

Routing / Message Broker – If you happen to need one, the throughput capabilities of your broker forms a floor on system performance. Try to make certain it will scale linearly, either with multiple brokers, or preferably through peer-to-peer distribution of the brokering function. Message overhead shouldn't depend on routing complexity.

Re-formatting – As noted last month, most often re-formatting is done within the adapter, and this is the most scalable approach. Its even better if formats are remotely configurable and have low overhead.

Transformation – If requirements permit, distributed transformation is more efficient and scalable than hub-based transformation. This is true even if state dependent fan-in or fan-out is required (and therefore some degree of persistence) or the transformation rules change frequently, but a local "transformation engine" may be required. Watch out for XML overhead. Pay attention to rules repository efficiency: both storage and access can be bottlenecks.

Business process support – A business process engine must support high-performance process definition, redefinition, and process state update and interrogation. Like other components, process management is most scalable if distributed. Process definitions can be replicated from a central repository, thereby enabling local and efficient lookups for each process instance. A slow process engine will become a serious bottleneck and impede scalability.

However you select an EAI or e-business infrastructure, flexibility, scalability, and performance should be the primary concerns. The first step is acquiring an understanding of architectural implications. A second step is to obtain scalability and performance numbers from vendors and their customers. Third, consider your own scalability and performance benchmarks (some comments on how next month). Last but not least, test scalability and performance with a prototype application. Careful evaluation will enable you to make a choice that best supports the integrity of your enterprise – now and far into a successful future.