Approaches to Asynchronous Task Management

Facilitating the execution of job execution in enterprise systems can include any number of architectures and approaches. Successful adoption of any such system requires review of the specific features and implementation effort required in available solutions and known needs. This discussion will provide an overview to the selection approach and what decision points exist in the areas of various task/job needs, how the type of tasks required affects architecture decisions and the overall system environment that provides for these tasks and its characteristics.

In short if the need within the design or operation of a developed application requires asynchronous task queueing and management then some form of message queue (MQ) foundation is my recommended approach. It provides the necessary enqueue/dequeue, status and often network and execution components that complete a fully functional approach. The selection of an appropriate MQ is also dependent on the relevant operating system preference and as such this discussion won't sanction a named platform. The concepts illustrated are or should be available in technologies such as RabbitMQ, the MSMQ foundation or any number of stable JMS providers. Task management also isn't the exclusive domain of integratable MQ providers and often in the enterprise the role of existing operating schedulers and job systems often have a place if not as a target for the task queueing then as the exclusive scheduling means available. In those cases the design features below aren't comprehensively available but reliable operations and simplicity can make up for their absence.

The needs of any executed task should be appropriately modeled and known prior to their implementation in a MQ system. The ability for the system complete consistent, atomic, asynchronous tasks rests in specific environment design provisions. These provide for the execution of tasks with data structures to support various inherent details of the execution within the system to support publication and subscription of the different roles or elements within the transaction. Target location can drive these data needs too. This discussion includes some assumption of a distributed nature for the source application but details on target type, execution behavior and communication type are typically required regardless of the complexity of deployment. Most important though for task execution though is detailed and configurable standard and error output and its verbosity. Being able to track job status is not just a partial/total basis for transaction atomicity but also provides for the required application and operations/operator reporting on the status of the developed system.

The type of task execution can drive provider selection or simply the implementation. Types include batch processing: asynchronous execution of grouped tasks; integrated application tasks: events invoked through typical user interaction (commit logical record); adhoc: tasks invoked as part of some selection to complete a task specific to user input or scenario. The subsequent actions can require specific network protocol and behavior and these are often

target operating system or framework specific. In most cases prudent selections should include options for all of these types as they are derivative of the same basic structures.

The environment in which these tasks are executed, whether it be within the boundaries of a developed application or a remote target of such, should be considered in the discussion as well. Some frameworks can provide full security models with which administrative responsibilities can be delegated in a predefined view or fully functional means. Again the requirements drive whether that is required. Nonetheless some security model or privilege separation are features to investigate. The remote management of task targets, especially in a distributed application, is something to also consider. The execution of tasks within this system does not just require successful packaging,enqueue,dequeue and digestion but also the ability to gauge the health and viability of the target. The complexity of this need is dependent on some factors of execution. Tasks executing outside of some central queue or thread (remote execution via an agent) then appropriate instrumentation and metrics should exist at all task boundaries. If tasks are executed from some central means (application queries various data sources) then the state of the target beyond the queue will typically suffice. Finally any of these environments must be stable and maintainable both by developers but also release group/vendor. In most cases the selection of bespoke application messaging platforms comes with additional administrative risks. By selecting systems with the appropriate environment provisions the ability to provide stable, available messaging to any number of design patterns is possible.

The messaging provider approach discussed provides solutions for both cases of blocking actions and race conditions. Malformed transactions are possible but tend to be at the implementation of the frameworks within their outer boundaries. Both conditions can be alleviated through publish/subscribe models of the message services. The publish/subscribe function provides for consistent structures that provide for task assignment and separation through redundant subqueues and exhaustive address and status stratification throughout provider implementation. Of the two situations, blocking conditions are most likely as the nature of dynamic systems means the likelihood of over-execution likely. If not by unfortunate circumstance than normally through prudent simulated workload testing. In these cases the chaotic nature of the base system/environment during the upper thresholds of design needs can incur failure, even with the best designed transactional models.

In summary if an application requires the consistent delivery of task outcomes the recommendation is an implementation of the preferred messaging system. The ability to separate function, assignment and schedule within the MQ structures provide for the design of compliant transactions but additionally other inherently required mechanisms for valid operations. Selection of these platforms should not be singularly based on these functions or an option’s overall functionality also. As with the decision of any new technology the longterm viability of overall compatibility with existing technology and staff skillsets should be included. With this and the perspective of the design stakeholders an appropriate solution can be discerned.