

Process Orchestration Design Principles
and Best Practices
Jason Zhong
Senior Consulting Architect

1	Introduction.....	3
2	Implementation Considerations	3
2.1	SOA Application Design Analysis	3
2.1.1	Top level business process logic and partner interactions	4
2.1.2	Messages and Correlation Set.....	5
2.1.3	Human Workflow	6
2.1.4	Modularization and Reusability	7
2.1.5	Handling Faults and Exceptions	8
2.1.6	Handling Process Governance	9
2.1.7	Separation of Concerns	10
2.2	Best Practice for Designing Orchestration Flow	11
2.2.1	Process Development Cycle	11
2.2.2	Important Design Principles Using BPEL	12
2.3	Use Integrated Testing for Quality Control	13
2.3.1	Process Simulation.....	13
2.3.2	Use BUnit for Automated Testing	14
3	Process Management and Maintenance Consideration.....	14
4	Engine Configuration and Tuning Best Practice	15

1 Introduction

This paper discusses common issues and design considerations related to implementing a SOA (Service Oriented Architecture) project using the ActiveVOS process orchestration technology to create service enabled composite applications. The principles described here are based on best practices gained from proven real-world experiences and it applies to implementation projects regardless of the development methodology, scale of the project and complexity of the design.

The goal of this paper is to promote a set of best practices specific to SOA and process orchestration that would eventually result in lower risk, higher productivity, better agility and resiliency and faster to market time in the development process. Some specific issues addressed in this paper:

- Important design considerations are enumerated so risks can be handled early on and fewer surprises will be uncovered at later stage.
- Design principles discussed here should be used as guideline for development throughout the project life cycle and checked often. Superior flexibility and agility in design enables smooth evolution of the application and long life. In another word, start small and plan big.
- Quality assurance is an integral part of the development cycle and should be planed accordingly.
- Manageability and governance are integral parts of the design and should not be addressed as afterthought.

This paper does not try to prescribe a rigid approach or implementation framework because we do not believe that there is a unique and singular approach towards SOA implementation. SOA project life cycle should be managed and governed by the same set of software engineering methodologies and project management principles adopted by the performing organization as deemed appropriate to the delivery requirements, scale, criticality and complexity of the project. What we really want to highlight are the significant factors and aspects that are unique to a VOS based implementation. The content of this paper should serve as input to the various phases of a project.

The audiences of this paper include anyone who would be involved in the scoping, planning, design, development, testing, deploying and maintaining and managing phases of a project.

2 Implementation Considerations

2.1 SOA Application Design Analysis

The first steps towards designing an architecture for SOA based orchestration process is to analyze the business requirements of the project. The purpose of this analysis is to use a top-down analysis process to gain as much understanding as possible so that the business requirement can be translated into requirements of the initial design. The focus

here is interoperability and other integration concerns such as security, quality of services, etc.

For example if stage 1 of a project is to enable a desktop client application to connect to a business process via intranet then it is often tempted to forgo the need to place any security related token in the data model because it is not needed for the initial implementation. However if at a later stage a web based self-service client needs to be added to the system, then the data model may need to be revised to accommodate this new requirement – this may introduce costly changes to the already completed desktop client because the XML schema has changed. Proper analysis upfront can avoid this kind of issues by designing the data model with sufficient flexibility and extensibility.

The analysis can be performed based on the categories or topics covered in the next few sections.

2.1.1 Top level business process logic and partner interactions

The more important and usually also the first step in analysis is to identify the main business process flows and the kind of interaction between the processes and other disparate business services required in order to achieve the new functions. The result of this analysis should be a set of process models, a list of partner services and the types and details of the interaction (*synchronous* or *asynchronous*) associated with the partners. Depending on the complexity of the logic you may want to use some process modeling language to create the model. Some modeling languages and tools work better with ActiveVOS, for example BPMN (Business Process Modeling Notation) enables users to produce BPEL artifacts that can be imported into ActiveVOS and used as the base for the initial development.

Close attention should be paid when examining the partner services because orchestration is all about dealing with partners.

- *Is there a well-defined interface and data model that describes the partner's interface?* Sometime home grown application may never have had a well-defined interface (but it just worked) so an additional step may be needed to examine and define an interface for the partner.
- *Is the WSDL and XML schema definition for the interfaces available?* ActiveVOS allows import of interface artifacts from shared repository into individual orchestration project and these resources can be cached locally. Some time the XML schema definition is not readily available but XML instance documents are, then these instance documents can be used by ActiveVOS to create the appropriated schema definitions.
- *What are the main characteristics of the partner services? Are they short lived, atomic transactions that usually complete in seconds?* If so then synchronous interaction pattern is most appropriate. If not (let's say the partner represents a human task), then asynchronous communication pattern should be used.

- *Are these definitions fairly stable or it is anticipated that they will change during the course of implementation?* The fundamental advantage of loosely coupled application development model rests on the assumption that service interfaces are well-defined and remain stable. Sometime several components are being developed simultaneously and the interface does evolve over time because new business requirements are being addressed – this is in general an undesirable practice because it is very costly to manage this kind of changes. Things may break all over the place because someone has just replaced a base *string type* with an *enumeration type* in a schema. If this cannot be avoided then an effective mechanism and policies for managing these interface needs to be put in place:
 - Interface artifacts may be put into source code control system and versioned.
 - Use the resource change notification feature of ActiveVOS to automatically monitor changes. Deployment of these artifacts should be strictly managed through change process.
 - Close and effective team communication is key to success here – think of using co-location to reduce the delay and ambiguity in communication.
- *Will there be a need to bridge between different protocols and message encoding?* If yes, list the transport protocols/encoding and the applicable technology to support them:
 - ActiveVOS supports the following protocols/bindings for consumption and producing services out-of-box: *SOAP/HTTP/HTTPS, SOAP/JMS, Plain XML/JMS, REST, JAVA/J2EE* and *Email*.
 - Use the custom invoke and custom receive handler mechanism provided by ActiveVOS. With these features you can write Java class to handle the message transformation and transport protocol conversion anyways you want.
 - Use an Enterprise Service Bus solution together with ActiveVOS if you need to support a wide spectrum of transport protocols/bindings. ActiveVOS process engine can be embedded in an ESB as a service engine and have the ESB to handle the transport related issues. Your best choice is to engage the professional services team at Active Endpoints for this kind of integration.

2.1.2 Messages and Correlation Set

The next step in the analysis process is that for each business process in the design you need to identify the inbound and outbound messages. You need to understand the format and content of the messages and how to manipulate/transform the messages content to facilitate information sharing and exchange among the partners in the orchestration flow.

And if asynchronous communication is required between the process and partners there may be a need to identify a set of correlation data that can be used by the BPEL orchestration engine to route the callback messages to the intended recipients. These data will need to be defined for each of such callback scenario. Sometime this may be a simple identifier that can be extracted from an inbound messages, sometimes this may be

a combination of several piece of data (for example a process id and a sequence number). The BPEL terminology for these data fragment is called *property*. The BPEL terminology for the rules to extract these values from a particular message is called *property alias*. The terminology for a complete set of correlation data that can be used for correlating a single callback is called *correlation set*. Multiple *correlation sets* may be used in one process to handle callbacks from multiple partners. It will be beneficial if there are clearly understood prior to the development stage.

2.1.3 Human Workflow

If you know that humans will be involved at some point in a business process – maybe as part of exception handling procedure then a workflow may be added as an integral part of the business process. Generally the detailed definition of human workflow and activities is a complex issue that needs to be worked out by business users. However this does not necessarily mean that designing and developing of people activities in process orchestration must wait for the definition to be completed. This can be done while the human task definitions are being elaborated.

To understand how human activities can be incorporated in a process, you need to find out if there is already a workflow system in place in your organization that exposes the human workflow as services? If the answer is yes, then the human tasks can be invoked as a service. This is the most straightforward way of adding a human interface. The drawback of this approach is that the human task interface is not distinguishable from other machine based services and typically you are locked in a proprietary human task system that may not evolve well for all your future need.

A better approach is to design and implement human tasks using the ActiveVOS human work flow system which is based on the WS-HumanTask specification. This also works well with BPEL because ActiveVOS supports the BPEL4PEOPLE specification. This standard based technology allows implementation of human tasks in a vendor neutral way so your investment in knowledge and code are protected if you need to migrate the work related to human activities to a different vendor's implementation of human task system.

There are three aspects to the design of human tasks that need to be considered:

- Task definition – this covers the task specific interface of the task that specifies the input, output and fault data of a task, as well metadata used for managing the life cycle of a task as defined in WS-HumanTask. Examples of metadata are potential owners, administrators, deadlines and escalation actions, priority, data for task presentation, etc.
- People activity in orchestration flow. Based on BPEL4PEOPLE people activity is first-class citizen in a BPEL process and its definition needs to be incorporated in the process definition so the process engine and the human task system can work together seamlessly. This includes mapping of data (including attachments) from the process to the input and output of the human tasks and handling the potential faults resulted from human tasks.
- Task rendering customization. Because a task needs to be eventually presented to a human for processing using a client application, the rendering of task data is

important for efficiency reasons. ActiveVOS provides a default rendering of task data that allows raw data being displayed and output being sent back. This will be most likely not adequate for a real-world implementation thus customization is necessary. ActiveVOS provides several standards-based mechanisms for task client customization using standard technologies. Design of the task client interface is a usability issue that needs to be taken care of with the actual users.

2.1.4 Modularization and Reusability

One of the benefits of Service Oriented Architecture is that atomic business component can be wrapped as service and re-used over and over again in forming high level business functions. SOA makes it easy to accomplish this by standardizing the ways services interoperate with each other. And because ActiveVOS is built on SOA standards, it can both consume and produce services that conform to the Web Service standards with little effort. Furthermore ActiveVOS allows invocation of one process from another process without the overhead of SOAP encoding and message dispatch. And you can invoke a process as a sub-process which allows for the sub-process to participate fully as a part of the calling process, which enables exception and compensation handling across the process boundaries.

So how to decide when to use processes and sub-processes? In a bottom-up approach you can start by designing process for simple business function then expose it as Web Service therefore it can be re-used by other higher level business functions. In a top-down approach you can look at a high level business process and ask the questions if this process can be divided into smaller, more manageable and loosely coupled sub-processes. Sometimes it is not clear if it makes sense to modularize a process because all the effort involved in creating the additional interfaces until you have built some of the processes. The following are signs to watch for determining if modularization is needed:

- The process is extremely large and complex, and it becomes very difficult to navigate, understand, debug issues and track changes
- The process consists of many scopes or units of work, whose boundaries are very clear (but may not have been defined)
- There are many repetitive patterns/blocks in a process and they only differ in the content of the variable and messages that are used inside these blocks
- Many processes contain code that share the same logic and you have done a lot of copy and paste between them

If you decide to use process or sub-process do pay attention to the following

- Design the interface between the process or sub-process carefully so not to introduce unwanted restraint in a sub process
- Balance the need of performance and manageability vs. the need of reuse
- Take care of passing transaction context or coordination between processes – use sub-process whenever it is appropriate

2.1.5 Handling Faults and Exceptions

Exception handling is important to any business process because things will go wrong in the real world. Some of the exceptions occurring in business process can be predicted at design time, such as certain faults returned by an orchestrated service. Some exceptions are hard to anticipate until you actually run into them at run time, such as an inbound message that conforms to the message definition but violates one of the underlying business rules (for example, a social security number that starts with '000'), or a downed communication link to a partner service.

2.1.5.1 Handle Anticipated Exceptions

For the first type of exception, fault handling logic can be built in the process itself, as part of the complete solution. There are two parts to this – the first part is to catch the exception based on some identifier. Because fault in one process may have a ripple effect on other processes, it is crucial to identify and define these potential faults at the early stage of design, than you can add fault handling code gradually at a later stage.

BPEL provides a structured mechanism for exception handling and unit of work in a process can be organized into more granular unit called *scope*, for which discrete fault handler can be specified. BPEL enforces the integrity of the fault handling logic by propagating the faults from inner scope to its enclosing scopes and ActiveVOS extends this concept to sub-processes.

As part of exception handling, sometimes it is necessary to roll back changes that have already been committed in order to maintain the integrity and atomicity of a unit of work, in a way similar to rolling back an ACID transaction. Because business process tends to be long running and orchestrated services are loosely coupled and may cross organization boundaries, it is not always practical or desirable to hold changes to resources in an uncommitted state. BPEL provides a solution in the form of structured compensation handler, which allows for logic that explicitly rolls back changes. And BPEL maintains the consistency of this mechanism by traversing the compensation handler chain from outer scope to its enclosed scopes and ActiveVOS extends this concept to sub-processes. A key requirement of implementing compensation handler is to identify the partner interfaces for compensation.

2.1.5.2 Handle Runtime Exceptions

For exceptions that cannot be predicted until it happens (thus they cannot be handled in the process itself), the consequences may be severe and you may need to find ways to alleviate them in accordance to the business requirements. In order to be able to take actions you need to be alerted when such exception occurs. ActiveVOS provides runtime alert service that can be used to generate notification and trigger other actions when an unhandled exception occurs. ActiveVOS can also suspend such process rather than let it terminated – which may not be an acceptable situation.

The alert service can be used for recovery purpose as well, as it will receive the full details of the exception. More often it requires human involvement to correct the condition that causes the fault and conduct the recovery manually. For example a fault caused by a typo in a social security number may require manual correction of the error in the message variable and retry the failed step in the process. ActiveVOS provides an interface facilitating such recovery activities. Some people may consider this similar to what a human workflow would do – the important difference is that human task may be used in a business process to handle predictable exception conditions as part of the process flow, whereas the suspend-retry function is mostly suitable for runtime error recovery. However you can design a process with people activities and assign the process as the alert service so human task can be used to handle uncaught exceptions.

2.1.6 Handling Process Governance

Governance is an important aspect of any SOA projects. Enterprise policies and business requirements for governance need to be translated into concrete actionable items as part of the design exercise. The following are some of the common governance issue:

- Security – this may cover authentication, authorization, access control, confidentiality, integrity and etc. Analysis may result in the mandate of employing technologies and standards such as WS-Security (secure SOAP message), SAML (single sign-on), SSL, other WS-Policy Assertions (privacy, Quality of Service), etc.
- Resiliency – this may mean that process needs to be recoverable in the event of system/component failure, or with built-in redundancy that provides fail-over.
 - ActiveVOS provides support for WS-ReliableMessaging, which allows messages to be delivered reliably between process and partners in the presence of system or application failures. You can also design your system to use messaging technology that provides such reliability if WS-ReliableMessaging is not supported by partners.
 - ActiveVOS supports process persistence which is required for recovery and failover. Please be aware that persistence comes at the price of increased resource consumption and increased response time due to database related activities. ActiveVOS allows specifying a persistence policy on per process basis which should provide some flexibility in this regard.
 - ActiveVOS supports engine clustering configuration which provides failover capability if it is required.
- Compliance – companies doing business need to be in compliance with government laws and regulations such as Sarbanes Oxley or Basel II. And organization may have formulated policies that regulate how business process should behave based on best practice such as ITIL or ISO. One important aspect of compliance is to be able to continuously measure, control and audit business activities.
 - The requirement of logging and auditing of business processes is an integral part of the business requirements. ActiveVOS provides logging and process persistence capability for deployment and process execution

activities. Specify the logging and persistence policy to be consistent with the compliance requirements

- Critical business process may need to have high availability and service window may be small or non-existing. Therefore the ability to support upgrade of deployed process without interruption of the running instances may be critical. ActiveVOS provides a runtime version managements feature using versioning policy. A versioning policy for a process version can specify effective/expiration date of a new version, running instances disposition and retention period of completed/faulted process instances. Design these policies to be consistent with the compliance requirement.
- Monitor BPEL engine alerts – ActiveVOS can be configured to generate events when certain KPIs of the engine have reached predefined thresholds, for example if the faulted processes count has reached a certain value. Create a Monitor Alert Service process if you want to receive these engine events.
- Monitor process fault alerts – you can create an alert service process to receive events related to unhandled fault generated in processes.

2.1.7 Separation of Concerns

You may find that when considering the design of a business process there actually are two kinds of business related logic that will impact the design. One is directly related to how services should be orchestrated to achieve a business goal. For example order processing may required inventory check, payment service and shipping service. This logic usually is quite stable once it is spelled out and remains unchanged after it becomes operational. The second kind of logic often has to do with decisions that need to be made in an orchestration flow, things such as how to evaluate risks associated with an issuing an insurance policy, or how to apply discount based on certain criteria of a purchase. This type of data tends to change quite often during the lifespan of a business process because the market condition may dictate the change. What is more is that when changes like this need to happen, they need to happen fast – otherwise business bottom lines may be impacted adversatively because it cannot react to changes in market conditions in a timely manner. This means that the normal change control process required for deploying changes to a process may be too rigid for this purpose.

Because of these reasons, it is often desirable to separate the more volatile, business policy based logic from the more stable process orchestration logic. Instead of hard-coding these business policies into an orchestration flow, one can use business rules to author and manage these business policies as separate assets, and deploy the rules to the business process at runtime. The combination of using business rules and orchestration flows creates a more agile and adaptive solution.

In order to make the right decision regarding whether or not business rules is appropriate for your specific implementation scenario, the following criteria may be considered.

- Are there decision points in the process that will change more often than the orchestration flow itself? If the changes happen much often than the release cycle allowed, considering using business rules.
- Who will initiate these changes? Are they the architects who design the orchestration flow, or domain experts who do not know much about the process flow? If the answer is domain experts then consider utilizing a business rules management system (BRMS) to give the domain experts the tool to manage these business policies.
- How fast these changes need to take effect? BRMS will allow fast deployment of business policies changes outside of the normal release cycle.

Using business rules in process orchestration requires the adoption of business rules software and more effort is required to learn, design, develop and manage rules. You need to balance the need of separation of concerns against this overhead to make an appropriate design decision.

We have created reference implementation that demonstrates how to use Business Rules with ActiveVOS.

2.2 Best Practice for Designing Orchestration Flow

Once you have completed the overall analysis of the requirement, you should have all the necessary elements to start creating the architecture and the processes based on the requirements. And once you have a blueprint in place, you can proceed to develop the processes and the associated artifacts.

2.2.1 Process Development Cycle

The development of individual orchestration process usually involves a number of steps:

- Import and locate resources related to services that will be orchestrated. This sometime involves the creation of artifacts based on sample data.
- Create the orchestration flow based on process models created during the analysis stage
- Quality Control
 - Collect sample data for testing
 - Simulate the process when new functions are added to the process
 - Create automated unit test cases and test suits for integrated regression tests
 - You can use either a recorded simulation session, or the BUnit wizard to generate test cases.
 - Test cases can be aggregated as test suits and test suites can be incorporated in the QA process
- Deployment
 - Create deployment artifacts. This is the process of defining endpoint reference and other attributes for the partners and the process itself

- Define the invoke handler and endpoint reference for the partner roles. Invoke handler specify what binding and protocols to send messages to the partners.
- Define the receive handler and endpoint reference for my roles. This defines how the service exposed by this process can be accessed by clients.
- Define policies related to my role and partner roles – this should address any security, communication and QoS attributes required for the defined interaction.
- Define process governance related properties – this includes versioning policies, persistence policies and runtime process management related data (indexed properties).
- Deploy and use the remote debugging capability of the designer to diagnose any runtime issues

2.2.2 Important Design Principles Using BPEL

Below are some of the most important considerations when you design a process orchestration using BPEL

- Robust – you want the process to perform as expected in all possible real world scenarios
 - Use the appropriated interaction pattern (synchronous vs. asynchronous). Synchronous communication is only suitable for low latency, short lived service invocation. Asynchronous communication can handle long lived process and interaction patterns but it requires more programming effort.
 - Use scope and isolated scope to separate the units of work in a process. Fault and compensation handlers can be attached to a scope so it is important to place the activities belonging to one unit of work into one scope so the integrity of the process can be maintained if some fault has occurred. If you have parallel activities and sharing resource between them, then use isolated scopes handle access to the shared resource.
 - Use structured fault and compensation handling to handle all known fault and exception conditions. If there is known faults that can be returned by a partner, then you should think how to handle it.
 - Use message validation to validate inbound and outbound messages whenever it is appropriate. Catching invalid message early and handle it can help to make the process much more robust. However be aware that validating message adds load to the runtime engine.
- Flexible – you want to create interfaces, data model and deployment artifacts that can be easily modified or extended at a later stage if required
 - Use extension mechanism in interface design to add flexibility. For example you can use XML schema anyType to indicate that a particular definition is extendable.
 - Use URN mapping to add flexibility to partner role binding. You can add a level of indirection when specifying the endpoint address of a partner when deploying a process. You can use tokens in the endpoint address

when creating the process deployment descriptor and substitute the tokens with components (host, port, service names, etc.) of the real address using the engine administration console. This eliminates the need of redeploying a process for the sole reason that a partner's endpoint address has changed.

- Efficient – you want the process to perform efficiently at runtime
 - BPEL contains constructs that allows for parallel execution. *Flows* and parallel *forEach* activity provide powerful semantics to express the attributes to execute the enclosed activities in a parallel manner. ActiveVOS supports true parallel execution using multiple threads for receiving inbound messages and invoking external services.
- Compliant – you want to create services that confirms to the most accepted interoperability standards
 - Use WS-I Basic Profile 1.0 as binding guidance when exposing process as services.
 - Use doc-lit style whenever possible so the message payload can be validated against schema
 - Use single part messages whenever possible
 - Do not use RPC-Encoded binding style unless you must. Encoded message cannot be easily validated.
 - When to use ActiveBPEL extension – ActiveBPEL provides extension to the features defined in the BPEL specification to address custom needs. Use BPEL extension with caution because this makes the process less portable. Below is a summary of the extension:
 - *Suspend, Continue, Break* - these permit logically driving a process into the Suspend state and more fine grained control of your looping constructs
 - XQuery, JavaScript as expression language, in addition to XPATH which is mandated by the BPEL specification
 - ActiveBPEL functions – these are functions (getProcessId, get Attachment, etc.) that provide access to features not covered by BPEL specification.
 - Custom invoke, custom function – this provides user a way to create Java based components and use them in expressions or as partner services

2.3 Use Integrated Testing for Quality Control

To ensure the quality of the application and reduce maintenance cost, ActiveVOS has a set of built-in features that makes it much easier to perform quality control. There features are simulation and BUnit test.

2.3.1 Process Simulation

Simulation allows a developer to test the correctness of the orchestration logic in the development environment. By providing sample data for all inbound messages you can created a “scenario” that simulates a set of business conditions. You can then run the

simulation against the process and check if the process behaves as expected under these conditions.

Simulation is a powerful tool to validate the process without the need of ever deploying it. You can simulate as many scenarios as you want for a given process. Simulation is especially useful for testing newly added logic. The key to simulation is to collect sample data early and understand how they can be applied for the test scenarios.

2.3.2 Use BUnit for Automated Testing

While simulation is good in validating new functionalities, you would have to run many simulations in order to have a good coverage of a process if it has multiple execution routes. And simulation requires human interaction which means you cannot automate it.

To solve the problem of test coverage (especially important for regression test) and test automation, you can use the BUnit test function provided by ActiveVOS.

BUnit test is very similar to JUnit in that it is driven by scripts, can be run outside of the IDE as part of a build process and does not require human interaction.

There are two ways to generate the BUnit test cases:

- Record a simulation session and save the result as a test case. Assertions are automatically to check the correctness of outbound messages.
- Use the BUnit wizard to generate a test case and manually modify the inbound message content and assertions. This is more suitable for advanced use case.

In most cases you want to bundle all test cases for a given process in a test suite so it can be incorporated into one offline test.

3 Process Management and Maintenance Consideration

There are a number of runtime configurable features of the ActiveBPEL engine that are related to process governance aspect of the business process. You need to be aware of these features so you can create the best runtime environment to meet the governance policies.

- Handle uncaught exception – this feature enables a process with uncaught (unhandled) exception to be suspended instead of fault-out. You can either manually recover the process or you can assign an alert (BPEL) service that will handle the situation automatically.
- Specify the logging preference – there are three levels of logging provided in the ActiveBPEL runtime environment. Process deployment log provides trace of process deployment activity. Process persistence provides a detailed description of the state of a process instance. Process log provides a chronicle of all the activities of a process instance. All logs when turned on are preserved in the database. Turn on the logs based on the logging and auditing requirements of the business process. Deployment log is always on. Process log can be turned on and

off for the engine. Process persistence can be turned on or off for individual process. Please note that enable logging will trigger additional overhead at runtime.

- Use engine monitoring feature – key performance indicators such as faulted process count, count of discarded inbound messages can be monitored using the engine monitoring service. You turn on the individual indicators and assign a BPEL process as the handler of the monitoring events.

4 Engine Configuration and Tuning Best Practice

In order to meet the requirement of performance, scalability and reliability the ActiveBPEL engine runtime environment needs to be properly configured and tuned, as well as the hardware platforms and 3rd party software components (such as DBMS, application server, etc.) the engine relies on. To learn more about the best practice related to engine deployment planning, configuration and tuning, please contact Active Endpoints directly for documents related to engine configuration and tuning.