# Table of Contents

- Who Should Read This Paper? ................................................................. 3
- What is the SBM Orchestration Engine .................................................. 3
- Scalability From Planning to Deployment ............................................... 4
- Planning for Scalability ........................................................................ 5
- Designing for Scalability ...................................................................... 7
- Deploying Scalable Orchestration Workflows ....................................... 9
- Other Best Practices During Design Phase .......................................... 12
- Summary ............................................................................................. 13
Who Should Read This Paper?

This paper provides guidelines on how to optimize orchestration workflows in Serena® Business Manager to maximize performance and scalability. It includes a brief overview of the SBM Orchestration Engine components, discusses different types of orchestration workflows, sets forth best practices for designing orchestrations and describes hardware and configuration approaches for making enterprise systems scalable.

Target audiences of this paper include technical architects, solution designers, and advanced designers.

What is the SBM Orchestration Engine

The SBM Orchestration Engine is a server that executes Business Process Execution Language (BPEL) instructions to coordinate—or "orchestrate"—Web service calls to create an SOA process. It is implemented using an open source BPEL engine, JBPM, distributed by JBoss, Inc. Serena has customized JBPM to make it more scalable and feature rich. For example, it has been modified to support throttling to handle enterprise scale loads, dynamic endpoints, and soap headers. It is also modified to work with the Application Administrator, Event Manager (EM), and SBM Application Engine.

Two types of orchestration workflows can run on the orchestration engine:

- **Asynchronous Orchestration Workflows** – These workflows are event based, which means the initiating system continues immediately after raising the event. An Event can be raised from the SBM Application Engine on transitions, from external systems using the Event Manager Web service API, by e-mail or using the JMS queue. Because requests are queued, SBM Orchestration Engine processing is decoupled from the initiating system.

- **Synchronous Orchestration Workflows** – This type of workflow can only be called from the SBM Application Engine. Unlike asynchronous orchestration workflows, the request is not queued, so the workflow is executed immediately and can return data like a Web service. The human process running on the SBM Application Engine waits until the workflow has completed and may use the return values to set item data.
The following diagram depicts the component architecture of the SBM Orchestration Engine:

![Diagram of SBM Orchestration Engine](image)

Scalability From Planning to Deployment

Creating a scalable enterprise system begins with requirements gathering and continues through development and final deployment. This paper discusses scalability issues in each of the following phases of the development lifecycle:

- **Planning** – In this phase, you create a high level definition of the process you would like to implement. You identify the information the process will collect, the systems and human workflows that will interact with the process, and any business logic needed to implement the process. In general terms, you should be defining requirements for an 'orchestration' of human workflows and Web services that brings people and system processes together. This paper will help you identify which types of processes should and should not be implemented using orchestration workflows. Generally, you should use orchestrations only to enable collaboration between systems and people. See Planning for Scalability [page 5] to determine if the function you need is best implemented using orchestration workflows.

- **Design** – In the design phase, you use SBM Composer to implement the requirements identified during the planning phase of development. Serena recommends using the 'path to production' strategy while designing process apps. In this approach, designers work in a separate development environment and get their process apps working there first. When the process app is functionally complete, it is moved to a testing environment where it is tested and tuned for performance. Finally, when it has been proven to be fully functional, stable and scalable, it is moved to a production environment for use by the enterprise. While it is acceptable to focus on meeting functional requirements during initial development, it is best to start thinking about scalability as early in the process as possible. See Designing for Scalability...
for methods that designers can use to create orchestration workflows to make them more scalable.

- **Deployment** – Once the design phase is complete and you have a working process app, it is time to seriously evaluate the load your system will bear under production usage. Aspects of this evaluation that you need to consider include:
  - The number of orchestration requests coming in during a given period of time.
  - The interval between orchestration workflow requests.
  - Any concurrency requirements of these requests.
  - The load on databases during Web service calls, including the number of connections requested at a time.
  - The computational load the orchestration places on the system.

The impact of many of these factors can be determined by testing the mashup and evaluating its performance as the application is exercised. Deploying Scalable Orchestration Workflows [page 9] focuses on strategies such as vertical and horizontal scaling as well as throttling that you can use to address scalability problems revealed during these staging tests.

### Planning for Scalability

The first decision you make when planning an application is to determine the best technology for implementing it. This section will help you determine whether or not orchestration workflows are an appropriate technology given the functional and scaling needs of your application.

**WHEN TO USE ORCHESTRATION WORKFLOWS**

Today’s enterprise applications typically require that distinct heterogeneous systems collaborate with each other to solve business problems. As more systems are brought together, the amount of communication and number of transactions required to achieve this goal has grown, putting emphasis on highly performing, scalable systems. The SBM Orchestration Engine meets these needs by enabling human workflows to collaborate with both existing legacy systems and new technologies.

Business problems that can be solved by the SBM Orchestration Engine include:

- **Collaboration with existing legacy systems** – SBM is designed to empower you to build the next generation of process apps to bring people and processes together. Orchestration workflows provide the integration point between SBM and legacy systems and can be initiated from either. Legacy systems without SOA capabilities can interact with the human workflow by sending specially constructed e-mails containing SOAP messages to fire off asynchronous orchestration workflows which interact with the SBM Application Engine. The Event Manager accepts e-mail events in multiple formats as shown in What is the SBM Orchestration Engine [page 3]. If a legacy system can receive Web service calls, they can be called from synchronous and asynchronous orchestrations executed during transitions.

- **Collaboration with SOA capable systems** – Human workflows can be easily integrated with other SOA capable systems that are prevalent in today’s workplaces. For example, Serena has published numerous process apps that serve horizontal and vertical industry demands. These process apps offer integrations with HR systems such as SAP, incident management system such as Remedy, and source control systems such as Subversion and Team Foundation Server – Visual Studio Team Systems. The integrations use orchestrations initiated from the
human workflow to call Web services for passing data to and receiving responses from the external systems. In addition, because the Event Manager can be called as a Web service, asynchronous orchestrations can be initiated from events occurring in the external systems to communicate back to the SBM application.

- **Intelligent data enrichment** – People using SBM human workflows make decisions and take actions based on information available within that system. But often, they can make better decisions if they have access to related information stored elsewhere. Intelligent data enrichment is the practice of supplementing information presented in a human workflow with relevant external data. While rich interface tools like the REST Widget can be used for simple integrations, more complex integrations require the use of orchestrations. A synchronous orchestration can implement business logic to collect and coordinate data from several sources for presentation in a form. For example, Serena’s Sales Credit Approval process app enriches the forms with the customer’s credit information. The credit information is gathered by a synchronous orchestration workflow using business specific logic.

- **Validation** – Data validation is another common and appropriate usage of orchestrations. While on-premise SBM application designers have typically used scripts to validate input, this capability is not available in the on-demand version. Synchronous orchestrations that use programming constructs such as comparisons, string and number manipulation, and loops can be used for the validation of user-entered data. In addition, data validation tasks can occur through the use of JavaScript in custom forms.

In all of these cases, the underlying purpose is typically collaboration involving multiple applications using business granular services rather than data processing. The orchestrations are in the context of Web applications with which people are interacting.

As with all integrations, you should evaluate your requirements for scalability and performance characteristics before committing to an implementation using orchestrations.

**WHEN TO AVOID USING ORCHESTRATION WORKFLOWS**

Although orchestration workflows are useful for performing many types of tasks, there are some types of work for which they are ill-suited. In this section, examples of some of these tasks are provided, along with explanations of why performance and scalability concerns would disqualify them for implementation using SOA and BPEL.

- **Visual Programming** – SBM Composer enables you to visually create orchestration workflows using number of BPEL constructs such as loops, decisions, data manipulation, and fault handling familiar to programmers. While it is possible to use these constructs as a general purpose programming language to create complex procedural programs, doing so leads to unsatisfactory results. These constructs are best used to apply business logic while coordinating or orchestrating the Web services called in the workflow.

  Drawbacks of adding long modules within orchestration workflows include the following:
  - Orchestration workflows become too long; they are very cumbersome to view, debug, and maintain.
  - Designers lose the benefit of compile time checks that other languages offer (such as Java).
  - Performance of these BPEL constructs is nowhere near as fast as that of traditional programming languages.
Complex logic or data manipulation should be moved out of the orchestration workflows and into Web services which can be invoked as part of the orchestration workflow. Doing this allows the workflow to ‘orchestrate’ Web services by applying simple and understandable business logic to the input data and the results of Web service calls. In this case, fault and compensation handlers can be correspondingly simple. On the other hand, if the workflow contains complex logic and data manipulation, then any compensation handler for that section of the process will be equally complex and unwieldy.

- **ETL Processing** – Data warehousing Extract, Transform, and Load (ETL) processing is a great example of when not to use an orchestration workflow. For this type of data intensive application, you want minimum per-record overhead and very fast data transformation processing. Implementing this using an orchestration entails at least two Web service calls per record that is processed, as well as a per-record overhead of the data transformation cost as implemented in BPEL. For all but the most trivial datasets, performance would be poor and the visual programming required would be unacceptably complex.

- **Synchronization** – There are two ways to achieve data synchronization:
  - In one scenario, synchronization takes place after the fact in a batch process. It involves processing a large number of records in multiple systems to ensure that they remain synchronized after changes to one or more of the data sets. Some examples of data synchronization are:
    - Synchronizing a customer or product master database with data stored in distributed systems.
    - Using orchestrations to synchronize data between two different disconnected systems after the fact (for example, synchronizing orders taken on a CRM system with a back end ERP system).
    - Using orchestrations to replicate data from one system to another and keeping them in sync. It is more appropriate to use orchestrations to connect the two systems in one process and avoid replication of data.

To achieve these using an orchestration workflow, you would need to iterate through each record on each system, making multiple Web service calls for each iteration, and then compare the data between systems and, if necessary, update the data by making additional Web service calls. For the same reasons cited above, this would lead to unacceptably poor performance.

In each of these examples, orchestration workflows are being used to do tasks best suited for implementation using a traditional programming language or specialized systems. The amount of data being processed is large and the number of associated Web service calls is too high. The complexity of some of the processes is not appropriate for the kind of visual programming supported by SBM Composer. In some cases, the work steps performed in the workflow are more granular than the business logic that represents the best use case for implementation using BPEL orchestrations.

- The other scenario of data synchronization is more process level and is a well-suited candidate for orchestration workflows. In that scenario, the data between two or more systems is continuously synchronized as the items flow through various business processes.

**Designing for Scalability**

Well-designed orchestration workflows are concise, well-annotated, and partitioned in logical groups. Each such group in these workflows should represent a business function in a process and handle
faults graciously in the event of application failures. Designing an orchestration with these practices in mind is key to achieving a scalable system.

The following considerations guide you to design a scalable orchestration workflow:

- **Business granular operations** – Many Web services offer both core object model operations and higher level business operations. For example, a Web service may have both an object model ‘CreateItem’ operation to create an item in the database and also a business level operation ‘CreateReservationRequest’ which internally calls ‘CreateItem’ for air ticket, hotel, and car. When possible, orchestration workflows should call operations that have business granularity.

- **Group operations** – Many Web services offer group operations. For example, a Web service may have ‘CreateReservation’ but it may also have ‘CreateReservations’ which takes an array of reservations. Because Web service calls are expensive, when possible, orchestration workflows should use group operations instead of calling the same operation in a loop. Each Web service call has a high performance overhead because the system must convert the Web service input and output messages and send them as XML over the network.

- **Avoid using the calculate step to transform data** – The expression editor in the SBM Composer calculate step exposes several data manipulation functions such as number, string, and substring. However, it is important to remember that the purpose of the calculate step is to assist in constructing Web service input messages. The underlying BPEL engine is not designed to efficiently transform large amounts of data.

- **Choosing fields in the orchestration link** – You can improve the performance of both asynchronous and synchronous workflows by limiting the data that is passed to the workflow. By default, all fields in the primary table are passed to an asynchronous workflow, but you can reduce that by deselecting any unnecessary fields in the Orchestration Link in the application. For synchronous workflows, you can pick both inputs and outputs of the workflow in the Orchestration Link. While reducing the amount of data passed to the orchestration workflow can improve the performance, you should not do so if you would subsequently need to retrieve the data by using a Web service call. The overhead of the additional Web service call is greater than any benefit gained by reducing the size of the inputs.

- **Pagination** – If a Web service returns a large amount of data in the response, the entire body of data is loaded into the JBoss memory. As a result, you may run into “out of memory” errors, which have the potential to crash the server completely. If you expect a large amount of data in the response, you need to design that part of the flow carefully. Instead of requesting the entire body of data in a single call, consider breaking it into smaller pieces. For example, instead of calling ‘getInfo’ on 100 items, you could create a loop that gets the information for twenty items at a time. Serena also provides throttling on payload size to avoid the problem of crashing the server when an unexpectedly large payload is returned by a Web service call. This is described below in Throttling [page 12].

- **Catch branches** – If a Web service throws a number of SOAP faults, your orchestration workflow should have separate handlers for each SOAP fault. Orchestration workflows allow you to inspect the fault data sent back from a Web service. Handling them individually reduces the complexity of the fault handlers and permits you to make your design clean and concise. In addition to being good engineering practice, it reduces the size of the workflow which in turn makes it more scalable.

By following these guidelines when you design your orchestration workflows, you can prepare for successful staging and performance and scalability tuning.
Deploying Scalable Orchestration Workflows

In this section, we discuss the scalable deployment of orchestration workflows. For information on scalability of other components, please refer to the "Scaling for the Enterprise" white paper.

Enterprise class applications must scale to meet the requirements of their user base. Successful deployment requires that you understand the needs of the application and its users as well as the limitations of the server resources you apply. For example, the JBoss server hosting the orchestration engine has restrictions imposed by the JVM it runs on. Under load, the server can manifest problems resulting from resource contention such as slow performance due to excessive CPU consumption, out of memory errors, or stack overflow errors.

Applications need to scale well in order to be successfully deployed in enterprise environments. This is a complex task involving many factors: network hardware and software, WAN latencies, server hardware and software, network load, server load and data volume. In addition, each type of application activity utilizes resources in a unique way. For example, a process app with many human workflow interactions will consume more resources in the SBM Application Engine, whereas a process app with complex orchestrations will require more resources to be allocated to Serena Common JBoss.

The following factors must be carefully considered by IT professionals along with Professional Services when planning a scalable deployment:

- **Network configuration/topology** – Serena Business Manager is distributed across clients, servers, and one or more database servers. Consequently, network hardware and software configuration and capabilities have a significant impact on its performance characteristics. The closer in proximity on the network these pieces are the better the performance. As a general rule of thumb, the SBM Server and SBM Database should be located on the same network segment.

- **CPU speed** – Server and database transactions are CPU intensive. Faster CPUs translate directly to improved server performance. Do not under-invest in the quality and speed of the server CPUs. The more processes that are running on a single machine, the higher the CPU demand will be once additional load is applied.

- **Software Configuration** – Serena Business Manager is a highly configurable product. While this is one of the product's greatest advantages, it can lead to performance issues if not configured in accordance with best practices. Support and Professional Services are well versed in these best practices and should be engaged to review implementations and help diagnose performance issues.

- **Usage Model** – SBM Composer provides users with the ability to create a wide variety of process apps. Serena Business Manager is used by different customers to solve different types of business problems. There are many possible usage models and each one will utilize systems resources in a unique way. Understanding the usage model of the user base will help in determining how best to scale the servers.

We use standard Microsoft and J2EE technologies and approaches for building and deploying a scalable enterprise solution. The Windows 2003 Server and Windows 2008 Server platforms provide industry-leading capabilities to manage and grow enterprise application performance and availability in an economical way. The price and performance of Microsoft SQL Server and Windows 2003/2008-based servers are consistently among the best as measured by the Transaction Processing Performance Council (TPC; reference [http://www.tpc.org](http://www.tpc.org)).
VERTICAL SCALING

Vertical scaling (or scaling up) involves applying higher powered hardware on the SBM servers. This can involve faster CPUs, multiple CPUs, more memory, faster network cards, or some combination of these.

- **Processors** – The SBM Server can take advantage of additional processors to improve overall performance. Using multiple processors enables you to scale your system vertically, provided your system has the capability to upgrade or expand the number of processors. For smaller installations in which all the SBM components are installed on a single server, you will find that implementing more processors enables your server to handle a higher work load.

- **Networking** – System networking is an important aspect of scaling. If the network card is saturated with network traffic in your configuration, you can add an additional network card to help scale the application. It is highly recommended that you place the SBM Server and Database server on a single subnet that has low latency.

- **Disk Subsystem** – The disk subsystem in a single system configuration can also be used to get additional scalability. The faster the disk drive speed, the faster disk I/O activity. Separating database data and logs onto separate physical drives (or spindles) will allow the system to operate more efficiently. If possible, the operating system should be placed on a drive of its own. Reducing contention for data access from the physical disk drive will improve the scalability of the system.


  On a 32-bit Windows 2003, the amount of memory that can be utilized by a single process is limited to 2GB. The amount of memory your system is able to address is partially dependent on the version of Windows 2003 server that you have implemented. You may find it necessary to upgrade your version of Windows 2003 server if you wish to scale the system memory beyond 4GB. (2GB of memory per processor is recommended.)

  For 32-bit Windows 2003 systems, if you wish to expand the SBM Application Engine memory utilization beyond 2GB of RAM, you can set an IIS switch (see [http://technet.microsoft.com/en-us/library/bb124810.aspx](http://technet.microsoft.com/en-us/library/bb124810.aspx)). This setting enables the SBM Application Engine to utilize up to 3GB of RAM. Serena Common JBOSS server can also take advantage of additional memory up to 2GB by changing configuration settings. Search the Knowledgebase at [serena.com](http://serena.com) for more information about changing the amount of memory available to JBoss. The SBM Database server can also benefit from additional available memory. Please consult your database documentation for further details.

HORIZONTAL SCALING

To achieve the highest levels of concurrency, scale your servers horizontally by moving some components to separate machines. Since the SBM Server and database processes contend for the most CPU and disk I/O, you should separate these components first.

You can scale the SBM Application Engine even further by implementing a Web farm that contains multiple IIS servers or JBoss clusters using either hardware or software. Hardware Web farms tend to be more costly to implement because they require you to have unique skills and specialized load-balancing hardware. The network load balancing capabilities of these Windows servers automatically distribute the load across the servers in the farm. A Web farm implementation is also transparent to end users—once the Web farm is implemented, users will continue to access the system via a single
URL. Using a load-balancer also provides failover capabilities. In the event that one of the IIS servers becomes unavailable, the load-balancer redirects incoming requests to other available servers.

For deployments with complex or high volume orchestration activity, additional JBoss servers can be configured. You can control the amount of activity that is directed to any one JBoss server by specifying particular end-points during the process app deployment process.

While all nodes in the cluster obtain the orchestration workflow definitions from same database, they execute workflow requests independently. At the heart of the orchestration engine cluster is the high availability JBoss JMS provider which provides load balancing and node failover services. When a request comes in to the master node, it is dispatched to the node with the lowest current load. If the master node fails, another node in the cluster assumes the role of the master node. This makes JMS message delivery highly reliable under even the heaviest loads. Asynchronous orchestration workflow requests always come to HA-JMS queue and can achieve high scalability.

On the other hand synchronous orchestration workflow requests always come from the SBM Application Engine and are not queued. To achieve higher scalability, external load balancer can be used as shown in figure 1.2. Load balancer determines which node in the cluster is best suited to execute the incoming request. To get more information on setting up load balancer, please contact Serena support.

The same method of load balancing can be used when there is a high volume of events coming into Event Manager.
THROTTLING

Even after scaling the enterprise system vertically and horizontally as described above, a system may encounter resource limitations under extreme load spikes. If you expect such spikes to occur in your orchestration usage, you can tune the throttling capabilities of SBM Orchestration Engine to manage the load.

There are three types of throttling supported by the orchestration engine:

- **Process counter based throttling** – By their nature, synchronous orchestration workflows are throttled by human activity on the SBM Application Engine. However, asynchronous orchestration workflows are triggered both by human activity as well as external systems using events. In this case, throttling controls the number of asynchronous orchestrations executing at any given time. Although the rate of execution is being throttled, all the events will ultimately be processed. The SBM Orchestration Engine runs an MBean to control the number of processes the engine can run at a time. When the counter goes beyond a set limit, the delivery of JMS messages is paused. When the number of processes falls below the limit, the delivery of JMS messages is resumed. This throttling of requests sent to the SBM Orchestration Engine can be configured to prevent them from exceeding resource limits. System administrators can configure throttling settings by running the SBM Configurator.

- **Payload based throttling** – When a Web service returns payload data, that XML data is loaded into the JBoss server memory. If the amount of data is excessively large, it can exceed the limits of the underlying operating system and JVM, causing out of memory errors. Because the amount of data returned cannot always be known during the design of the system, SBM offers payload throttling to address this problem at runtime. If the data size exceeds a preset threshold during the reading of the payload data, the read operation is stopped and the failure is logged in the Common Logger. This failure report is easily accessible to a system administrator. This feature is turned off by default for on-premise customers. If you anticipate that you may receive excessively large payloads from Web services not under your control, we strongly recommend that you turn this feature on.

  **Note:** Throttling is not available for synchronous orchestrations. The SBM Application Engine has a timeout of 30 seconds while invoking synchronous orchestration workflows. This value cannot be changed. If the orchestration workflow does not return in 30 seconds, the SBM Application Engine will report a connection failure error on the form.

Other Best Practices During Design Phase

As described in the ‘Designing for Scalability’ section above, there are many things to consider during the design phase to meet scalability goals. There are, however, a number of other best practices you should keep in mind during design that are not directly related to scalability. These best practices will help make your orchestration workflows robust, complete in functionality and maintainable.

- **CatchAll branch** – After adding catch branches for each specific Web service faults, you should use the ‘CatchAll’ branch to catch SBM Orchestration Engine generated faults. For example, if a Web service end point is incorrect, the SBM Orchestration Engine throws an ‘Invocation Failure’ fault which can be caught by ‘CatchAll’.

- **Logging / Notification** – You should consider writing your own logging or notification Web service. You can call this Web service in event of failures and to facilitate tracking and debugging of your orchestration workflow.
• **Compensation** – If your orchestration workflow affects the state of a database, for example, by creating or updating items, you should be sure to use compensation handlers appropriately. If a Web service call fails and raises a fault, the compensation handlers can rollback the database changes and return the system back to its original consistent state. For more information on compensation handling, please refer to the *SBM Composer Guide*. You should also remember to call 'compensate' in the sequence that will achieve the compensation result you need. By default, the BPEL engine calls compensation handlers on successfully completed scopes in reverse order. If you need to, you can change the order by explicitly calling compensate on the scopes.

• **WSDL limitations** – The *SBM Composer Guide* describes the current WSDL limitations of the SBM platform. You should understand which Web services are supported by consulting this guide early in the development effort.

**Summary**

This paper outlines a variety of approaches that collectively can be used to achieve highly robust and scalable orchestration workflows. At the planning stage, you should evaluate whether orchestration workflows are the right technology to use for implementing the feature you require. At the design stage, you should ensure that the orchestration handles the business logic you need without burdening it with high data volumes and granular Web service calls. Finally, at the deployment stage, you should consider the performance and scalability requirements of your system and allocate network and server resources to meet those requirements.

By following the guidelines set forth above, orchestrations can meet a wide variety of integration needs. If you would like help achieving this, Serena's Professional Services team has extensive experience in planning, designing, and deploying the most demanding enterprise systems.