



BMC Atrium Orchestrator Workflow Development Best Practice

A compilation of techniques, methodology, experience,
and research that promotes the successful development
of BMC Atrium Orchestrator workflow processes.

CONTENTS

INTRODUCTION. 1

BMC ATRIUM ORCHESTRATOR: AUTOMATION 1

BMC ATRIUM ORCHESTRATOR: WORKFLOW CONCEPTS. 2

REQUIREMENTS BEST PRACTICES 2

DESIGN BEST PRACTICES 3

 DESIGN WORKFLOWS WITH LOW COUPLING AND HIGH COHESION 3

 ENFORCE BEST PRACTICES. 3

 BUILD REUSABLE CODE WHEN POSSIBLE 3

 USE INCREMENTAL DEVELOPMENT. 4

 QUALITY CONTROL 4

 CHANGE CONTROL 4

 DEFINE WELL THOUGHT-OUT INTERFACES 4

 USE A TEST-DRIVEN DEVELOPMENT APPROACH 4

 STRESS TEST EARLY AND OFTEN 5

BMC ATRIUM ORCHESTRATOR: WORKFLOW CODING BEST PRACTICES 5

 NAMING CONVENTIONS. 5

 SOLUTION DEVELOPMENT STRATEGIES 9

 ADDITIONAL FACILITIES. 9

 DEBUGGING 12

 METRICS. 12

 TIPS. 13

CONCLUSION 13

APPENDIX A 14

 AUTOMATION OVERVIEW: WHAT DOES AUTOMATION MEAN? 14

INTRODUCTION

This document presents a compilation of techniques, methodology, experience, and research that promotes the successful development of BMC Atrium Orchestrator workflow processes. Some of the techniques and methodologies used in the development of workflows include:

1. An iterative, development process
2. A requirements management process
3. Quality control
4. Change control

Iterative development processes use an incremental methodology that progresses in incremental stages to maintain a focus on manageable tasks and ensure that earlier stages are successful before the later stages are attempted.

A requirements management process addresses the problem of creeping requirements, which is a situation in which the client requests additional changes to the product that are beyond the scope of what was originally planned. To guard against this common phenomenon, requirement management employs strategies, such as documentation of requirements, sign-offs, and methodologies, such as the use case.

Quality control is a strategy that defines objective measures for assessing quality throughout the development process in terms of the product's functionality, reliability, and performance.

Change control is a strategy that seeks to closely monitor changes throughout the iterative process to ensure that records are intact for changes that have been made and that unacceptable changes are not undertaken.

In the following pages, you will find a well-defined method that contributes to a successful step in the development of solutions using BMC Atrium Orchestrator workflow processes.

BMC ATRIUM ORCHESTRATOR: AUTOMATION

Automation in the BMC Atrium Orchestrator world is made up of workflow processes built with *activities* that may call other workflow processes. Workflow processes designed in BMC Atrium Orchestrator produce XML files that are based on BPEL (Business Process Execution Language) and are interpreted and executed by Peers (specifically, CDPs and APs) on a peer-to-peer grid.

Workflow processes can be configured to run on *this peer* (the one currently hosting the master Job Manager), *any peer* (assigned in a round-robin fashion), *best peer* (based on a simplistic evaluation model), or *named peer* (always run on the peer named "Big Peer," for example). The great majority of the time, workflow processes will be left at the default *any peer* setting.

Workflow processes, once begun, will execute to completion on the same peer. The process will terminate either with *success* (i.e. it makes it to the end without an unexpected error, although things might break in an "expected" way) or in *compensation*. A compensated state typically occurs when an adapter is incapable of executing (for example, the target address for a BMC Remedy Actor Adapter is invalid; the BMC Remedy server is down; or a running instance of the adapter cannot be found). Another frequent cause of compensation is not providing a value for a required input to a process (though these are usually caught during testing). Dividing by zero with the Utility Activity will also result in compensation. Another possibility is that the Peer on which the Activity was running fails in some way, such as a power failure, hard drive crash, etc. In this case, the system will automatically recognize that that job did not complete and will reassign the job to another peer.

Activities within a workflow process are usually assignment, flow-control, and decision-making steps, with one exception: call-adapter. The call-adapter activity, brokered through the Adapter Manager, might be executed on a different peer than the one on which the containing workflow process is running (i.e., a peer on which the appropriate adapter has been enabled). An example representing an *HTTP Get* can be found in Figure 1:

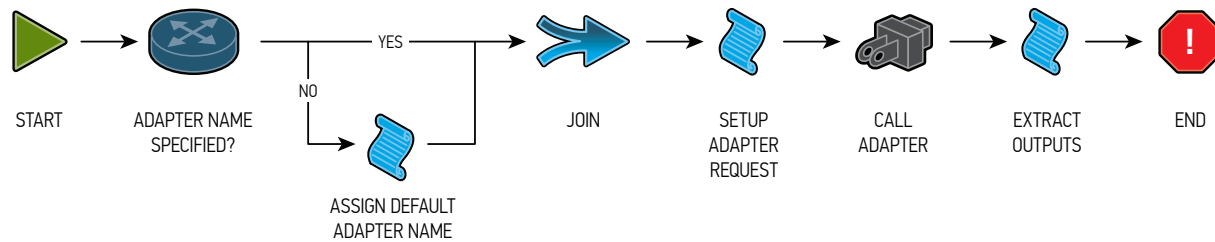


Figure 1. BMC Atrium Orchestrator Development Studio – Workflow Design.

The *Switch* and two initial *Assigns* happen on Peer A; the Call Adapter activity is interpreted on Peer A; but the requisite HTTP adapter is on Peer B. At this point, the process holds while the Adapter Manager delegates the actual HTTP request to Peer B's adapter. The request is made from Peer B (the HTTP server will see Peer B's address as the originator). The response is then sent back via the Grid framework to Peer A, which picks back up then moves to the final *Assign*. This is an important factor to consider when setting up firewalls or DMZs, or evaluating logs or other host data when executing a process.

BMC ATRIUM ORCHESTRATOR: WORKFLOW CONCEPTS

The following section introduces workflow concepts and how they apply to BMC Atrium Orchestrator workflow development. A workflow process is a collection of actions designed to produce a specific output from specific input when triggered by a specific event. Processing is accomplished through the execution order. In a workflow, a specific sequence of actions, when performed on specific data will yield a specific outcome.

A workflow process has the following attributes:

- » Specific events
- » Specific inputs
- » Specific artifacts
- » Actions created using activities
- » Execution order
- » Specific outputs

Events are objects that act as triggers or commands that capture the passing of control to the workflow. Events usually contain an entry point and a data source if one is necessary. Events are consumed by the workflow and are usually uniquely identified objects.

A workflow will process specific inputs to generate specific outputs. Inputs are consumed during the processing. Consumption may be in the form of the translation of input to output.

The specific artifacts come in the form of configuration items, environment variables, and templates. These resources are not consumed during the processing.

REQUIREMENTS BEST PRACTICES

Requirements management practices address the problem of creeping requirements, which is a situation in which a client requests additional changes to the product that are beyond the scope of what was originally planned. To guard against this common occurrence, requirements management employs such strategies as documentation of requirements, sign-offs, and methodologies, such as the use case.

Guidelines include:

1. Whenever possible, document the list of requirements that will be implemented in the solution. The requirements may be in the form of user stories with acceptance criteria.
2. Be prepared for — and plan for — requirements changes. Usually, because the flexibility and rapid development offered by the BMC Atrium Orchestrator platform, the initial requirements will change over the lifecycle of the solution development.
3. Reserve time to review and elaborate user stories.
4. Ensure user stories have acceptance criteria.
5. If possible, employ a QA resource to verify that acceptance criteria are indeed met.
6. Evolve QA test cases with the evolution of the requirements.

DESIGN BEST PRACTICES

DESIGN WORKFLOWS WITH LOW COUPLING AND HIGH COHESION

Coupling is a measure of how interrelated two components are, and *cohesion* is a measure of how related the functions performed by a workflow are. To achieve low coupling in BMC Atrium Orchestrator workflows, ensure that changes to any component of a workflow do not affect other components in that workflow. An example would be to isolate BMC Atrium Orchestrator adapter functions from compute or translation functions in the workflow. This would ensure that a change to the type or functionality of the BMC Atrium Orchestrator adapter would not affect the rest of the workflow. High cohesion is achieved by ensuring that a workflow is designed to perform only one function or many functions that are very similar. This would ensure that errors can easily be isolated and the workflow would be modular.

ENFORCE BEST PRACTICES

Designing BMC Atrium Orchestrator workflows, while adhering to established best practices, is enabled by the BMC Atrium Orchestrator platform architecture. BMC Atrium Orchestrator workflows are written/created/coded in BMC Atrium Orchestrator Development Studio, hereafter referred to as DevStudio, a designer tool executed on the BMC Atrium Orchestrator Grid. DevStudio presents a modular architecture that offers the facilities for a layered development of workflows. A module is a folder containing rules, schedules, configurations, workflows, and concepts, which are artifacts used in the creation of the workflows in the module.

BMC Atrium Orchestrator separates modules into *Projects*, *Applications*, *Operations Actions*, and *Adapters*. Project modules should house all global solution variables in the module configuration section. Global variables can be adapter configurations, access control and user data, credentials, and any other information that covers all the solution components. At the application module level, runbooks should be defined. Runbooks implement the business processes that are orchestrated by the solution. Examples of runbooks are *Continuous Compliance for Servers*, *Incident Enrichment*, and *Service Provisioning*.

Runbooks use the next layer, *Operations Actions*, to interact with those services being orchestrated. Operations Actions will model the services provided by the end system. In the case of a hyper-visor, the services offered can be *Create VM*, *Update VM*, *Delete VM*, *Power On VM*, etc. The Operations Actions use the Adapter Utilities Layer to access the target systems through adapter actions. BMC Atrium Orchestrator adapters interface to the end systems through the API provided by that system. Adapters usually provide the basic CRUD (create, read, update, and delete) operations and sometimes multiple “perform actions”.

Further abstraction and modularization can be achieved through the use of folders inside the module. Folders can be used to group workflows by function. Using this layered approach can provide a level of abstraction that makes it easy to maintain low cohesion and high coupling.

BUILD REUSABLE CODE WHEN POSSIBLE

By default, having loose coupling and high cohesion will result in reusable workflow code (code that is not tied one specific data source or that contains functions that are not related). Additionally, strive to produce workflow components that perform only one specific function that can be reused at several different points in the workflow.

USE INCREMENTAL DEVELOPMENT

In this approach, the development of the workflows progresses in incremental stages. This approach helps to maintain a focus on manageable tasks and ensures that earlier stages are successful before the later stages are attempted.

QUALITY CONTROL

Use quality control practices as a strategy that defines objective measures for assessing quality throughout the development process in terms of the solution's functionality, reliability, and performance.

CHANGE CONTROL

Employ a change control strategy that closely monitors changes throughout the iterative process to ensure that records are intact for changes that have been made and that unacceptable changes are not undertaken. This guards against the catastrophic failures of the development environment or the need to roll-back changes in requirements that turned out to be unnecessary or the result of flawed logic.

DEFINE WELL THOUGHT-OUT INTERFACES

Workflows in a solution usually interact with many different external systems. Well defined interfaces promote loose coupling and thus make your workflows easier to change and maintain. Some goals of your interface design should be:

1. If possible, add an index field for auditing and identification
2. If the access type is asynchronous, pass only the essential data that will be required to access additional data synchronously
3. Use meaningful field names
4. Standardize or simplify data before passing
5. Simplify data structures whenever possible
6. Add meta data for validation whenever possible

USE A TEST-DRIVEN DEVELOPMENT APPROACH

The following is an adaptation of the Test Driven Development (TDD) methodology. The adaptation is presented as a sequence of steps that make up the development method. This adaptation is based on the methods of Test-Driven Development that many consider to be the canonical source on the concept in its modern form.

Step 1: Add a test

In test-driven development, each new feature begins with writing a test. This test must inevitably fail because it is written before the feature has been implemented. (If it does not fail, then the proposed "new" feature is obviated.) To write a test, the developer must clearly understand the feature's specification and requirements. The developer can accomplish this through use cases and user stories that cover the requirements and exception conditions. This could also imply a variant, or modification, of an existing test. This is a differentiating feature of test-driven development versus writing unit tests after the code is written: It makes the developer focus on the requirements before writing the code, a subtle but important difference.

Step 2: Run all tests and see if the new one fails

This validates that the test harness is working correctly and that the new test does not mistakenly pass without requiring any new code. The new test should also fail for the expected reason. This step tests the test itself, in the negative: It rules out the possibility that the new test will always pass, and therefore be worthless.

Step 3: Write some code

The next step is to write some code that will cause the test to pass. The new code written at this stage will not be perfect and may, for example, pass the test in an inelegant way. That is acceptable because later steps will improve and refine the code. It is important that the code written is only designed to pass the test; no further (and therefore untested) functionality should be predicted and 'allowed for' at any stage.

Step 4: Run the automated tests and see them succeed

If all test cases now pass, the programmer can be confident that the code meets all the tested requirements. This is a good point from which to begin the final step of the cycle.

Step 5: Refactor code

Now the code can be cleaned up, as necessary. By re-running the test cases, the developer can be confident that refactoring is not damaging any existing functionality. The concept of removing duplication is an important aspect of any software design. In this case, however, it also applies to removing any duplication between the test code and the production code — for example magic numbers or strings that were repeated in both — in order to make the test pass in *Step 3: Write some code*.

Step 6: Repeat

Starting with another new test, the cycle is then repeated to push forward the functionality. The size of the steps should always be small — as few as 1 to 10 edits between each test run. If new code does not rapidly satisfy a new test, or other tests fail unexpectedly, the programmer should undo or revert in preference to excessive debugging.

Continuous integration helps by providing reversible checkpoints. When using external libraries, it is important not to make increments that are so small as to be effectively merely testing the library itself, unless there is some reason to believe that the library is buggy or is not sufficiently feature-complete to serve all the needs of the main program being written.

STRESS TEST EARLY AND OFTEN

Make it a practice to perform continuous stress testing. Continuous stress testing would ensure that you are aware of how your solution would perform under various levels of loading. BMC Atrium Orchestrator is an excellent platform for orchestrating the actions of many independent application systems through the use of its library of call-adapters. BMC Atrium Orchestrator uses incoming events and messages or outgoing instructions and commands. Stress testing will expose many race conditions and situations where messages may be delayed or dropped because the receiving or sending workflow component cannot respond in a timely manner.

BMC ATRIUM ORCHESTRATOR: WORKFLOW CODING BEST PRACTICES

The following is a collection of coding best practices, developed over time, from the experience gained from developing workflows for the BMC Atrium Orchestrator Platform and BMC-sponsored solutions. These practices take full advantage of the facilities provided by the BMC Atrium Orchestrator Platform and DevStudio to develop workflows with low coupling and high cohesion. The Best Practices first examine *Naming Conventions* followed by *Solution Development Strategies*.

NAMING CONVENTIONS

Solutions developed using BMC Atrium Orchestrator workflows will consist of one or more runbooks supported by OA modules and adapters. The names selected for modules, folders, processes, context items, and concepts should be chosen with care. The goal is to use names that are consistent within the module and meaningful to the task being accomplished. In all names chosen, the following should be followed as a guide:

Input and output parameter names:

1. All input and output parameters must have human readable names chosen to accurately describe the required data in the context of the function to be performed.
2. When a context item/variable is used in multiple workflows, the same name should be used across all workflows. Example: Use *incident description* in all workflows that require the use of that data. Additionally, DevStudio has the capability to match inputs to outputs if the names are the same in both workflows.
3. Whenever possible, do not use the shortened form of words. For example, use *number* instead of *num* and *continue* instead of *cont*.
4. Workflow inputs and outputs must have descriptions that include valid values when the input is restricted to a list of acceptable values and indicate if a default value will be used for optional inputs where a user does not provide an input.

Example:

```
input name \[Optional\]:  
description of input:  
valid values: a comma separated list of values  
default value:&nbsp;&nbsp; &nbsp;  &
```

- Inputs that are of type XML must include a sample of that XML that includes all the possible elements. If necessary, include some possible variations of the XML input.

Example:

```
<ip-addresses>
  <ip-address>
    <first-octet>10</first-octet>
    <second-octet>1</second-octet>
    <third-octet>1</third-octet>
    <fourth-octet>4</fourth-octet>
  <ip-address>
  <ip-address>
    <first-octet>10</first-octet>
    <second-octet>1</second-octet>
    <third-octet>1</third-octet>
    <fourth-octet>5</fourth-octet>
  <ip-address>
</ip-addresses>
```

6. Inputs that use the same value should have the same name for consistency within a module. For example, in the case of *host* and *host name*, if these are each expecting a host name or IP address, they should be called *host*. Similarly, *username* and *filename*, when used as Input/Output Parameter Names, should be *user name* and *file name*. This rule should be followed for all similar names.

7. Whenever possible, the order of defining workflow input parameter names should be in the same order. For example, consider two workflows *Add Host to Cluster* and *Remove Host from Cluster*.

- a. In *Add Host to Cluster* the first few inputs are:

- i. adapter name
- ii. version
- iii. alternate username
- iv. alternate password
- v. cluster compute resource inventory path
- vi. host name
- vii. _____

- b. In *Remove Host from Cluster*, the inputs are:

- i. adapter name
- ii. version
- iii. alternate username
- iv. alternate password
- v. host folder inventory path
- vi. host name

8. To be consistent, the common inputs in each process should be as listed first, as shown below, with the unique inputs for each process following:
 - a. adapter name
 - b. version
 - c. alternate username
 - d. alternate password
 - e. host name

Context items/variables

1. Context item/variable names should be all lowercase words. Multi-word names should use spaces instead of CamelCase or underscores.
2. Global context items and static context items should have the first letter capitalized, e.g. *World Cup Soccer* instead of *World_Cup_Soccer*.
3. The name should accurately describe the value it holds. For example, use *change id* instead of *index*.
4. Acronyms should always be in uppercase, in process names and input/output parameter names. Use *IP* instead of *Ip*, and *CPU* instead of *cpu* or *Cpu*, and *number of CPUs* instead of *num CPUs*.

Workflow internals

1. All switch activities must be labeled with a question that yield either a yes/no or some specific/computed value. Example: *File exists?* or *System type?*
2. All paths flowing from a switch must be labeled with a representation of the resultant condition.
3. Include a labeled default path.
4. All activities must have a process specific name and not the default name.

Module configuration

When creating node items in the Module Configuration, the format will be as follows:

1. The first letter will be capitalized.
2. Multi-word names will be separated by an underscore '_'.
3. When the module configuration refers to an application, the manufacturer and full application name must be used.
4. Restrict the use of module configuration elements to only the essential attributes needed to configure the module. This is due to the fact that all elements defined in the module configuration are held as global variables. Note: In the BMC Atrium Orchestrator platform, the use of global variables is very expensive. This is because whenever a global variable is accessed, it must be synchronized across the entire grid.

Module names

Historically, module names were generated by DevStudio as a two-part name linked together by *SA*, *OA* or *AD*, each of which instructs DevStudio of where to display the module in the project explorer. Including an "_" would result in a space being displayed in DevStudio.

Depending on the solution, module names should be chosen to give as much information as possible about its owner, contents, and function. For example, choose a name that identifies the vendor, solution, and function. The module name would be represented as Vendor-OA-Solution-Function.

Examples:

- » *AutoPilot OA Applications Utilities* contain operations actions utilities
- » *AutoPilot OA Change Management* contains task management functions
- » *VMWare OA Virtual Server Management* would contain reusable workflows for performing hypervisor actions.

Folder names

Folder names should represent the category of activities that is performed by the processes contained in the folder.

Examples:

- » Search will contain all processes that perform searches.
- » Standard OA folders are *Setup_Validation*, *Search*, *Custom*, and *Utilities*.

In multi-word folder names, use an underscore or, in a post 3.0 Studio environment, a space can be used. In Studio 3.0 the underscore will be displayed as a space in project explorer.

Process names

Top-level processes in any module should be composed of the base processes that implement the functionality of the solution being built. This base processes, at a minimum, should include *Create*, *Update*, and *Delete* actions. Base processes should be designed to take a small number of complex XML inputs and to produce a small number of complex outputs. These inputs/outputs may be categorized as follows:

- » Application details <items/>
- » Network access: <ConnectionDetails/>
- » Process options: <ProcessOptions/>

Along with these base processes, a series of helper-processes should be created to accept simple inputs that can construct the complex input required for the base processes. Processes should be created to accept simple input that will be transformed into the complex <items/>. In a similar fashion, host name, user name, port, etc. will be transformed into <ConnectionDetails/>.

Example:

- » Create, Find, Delete, Update will receive <items/> as input.
- » <items/> is constructed from input elements and/or other complex documents.

The processes that create the complex XML that will be used as input to base processes should be named *Construct item for <xml document name/>*. Examples are *Construct item for Connection Details* or *Construct item for Configuration Item*.

The names chosen should reflect the action being taken by the process. Examples of OA Utilities processes are: Create Object, Delete Object, Get Object, Put Object, and Perform Action (where Action consists of a verb and object). These processes are expected to perform one action on an external system.

Examples:

- » Get Routing Table
- » Perform Change Route
- » Perform Add Route
- » Delete Route
- » Create ARP Entry
- » Delete ARP Entry
- » Get ARP Table

Process descriptions

The process description should contain a concise description of the purpose and function of the process. Input and output descriptions should not be included in the process description.

Passing parameters between processes

Whenever possible, avoid using static XML as input to a transform or called process when other input parameters are available. Example: Don't use a value of <empty/> as the input to a transform where all other data is contained in tokens to the transform.

SOLUTION DEVELOPMENT STRATEGIES

Solutions created using BMC Atrium Orchestrator workflows can be abstracted and decomposed into manageable chunks by using the layers provided by the BMC Atrium Orchestrator Platform. The layers provided are the *Projects*, *Applications*, *Operations Actions*, and *Adapter Utilities*. Starting with the Use Case or User Story proceed as follows:

1. Write the user story in the following format.

```
As a <user>,  
I would like to <action>,  
so that <value>.
```

Example:

```
As a change manager,  
I want to approve the change for a given change id,  
so that I don't have to use the UI.
```

2. Next, elaborate the story with the system specifics. This is achieved by adding system and environment data to the story. The important idea here is to identify the steps needed to accomplish the task.

Example:

```
Connect to IT Service Management and authenticate as change manager, using the  
change id find the change and change its status to approve. Return a success if  
completed successfully.
```

3. Putting it together, you are left with a Project: *_ITSM Global Parameters* or some such name. This module details the global configurations.

Application: *ITSM Change Management Actions* This module will contain the workflows that implement the business processes.

The workflows are:

- » Authenticate User(target, username, password)
- » Get Change by Change ID(target, change id)
- » Extract Change Status(change id)
- » Update Change Status(change id)

4. The next step is to identify the Operations Actions and Adapter Utilities needed to complete the solution. For example, you *Get Change by Change ID* can be found in the *AutoPilot OA Change Management module*.

ADDITIONAL FACILITIES

Concepts

The BMC Atrium Orchestrator common data model structure consists of a collection of data models, each of which is called a *concept*. These concepts are used to store general data components that are mapped to any structured elements or vendor specific fields for interfacing with third-party applications. These mappings are held globally for the workflow.

Concepts are used in BMC Atrium Orchestrator to pass structured data. Structured data in BMC Atrium Orchestrator is represented as an XML document. Using concepts, structured data can be easily accessed and transformed using an assignment or simple XPath statements. DevStudio provides a concept editor that is accessible through the *Assign* activity. When used in this way, data stored in a concept can be easily accessed using an assignment. The following show the context item *class name* being assigned to the concept element *vm-system-identifier* of the concept *AutoPilot-OA-Configuration-Management-Configuration-Item*. In this example, the concept name is *Configuration-Item* and it is located in the *_concepts* folder of the *AutoPilot-OA-Configuration_Management module*.

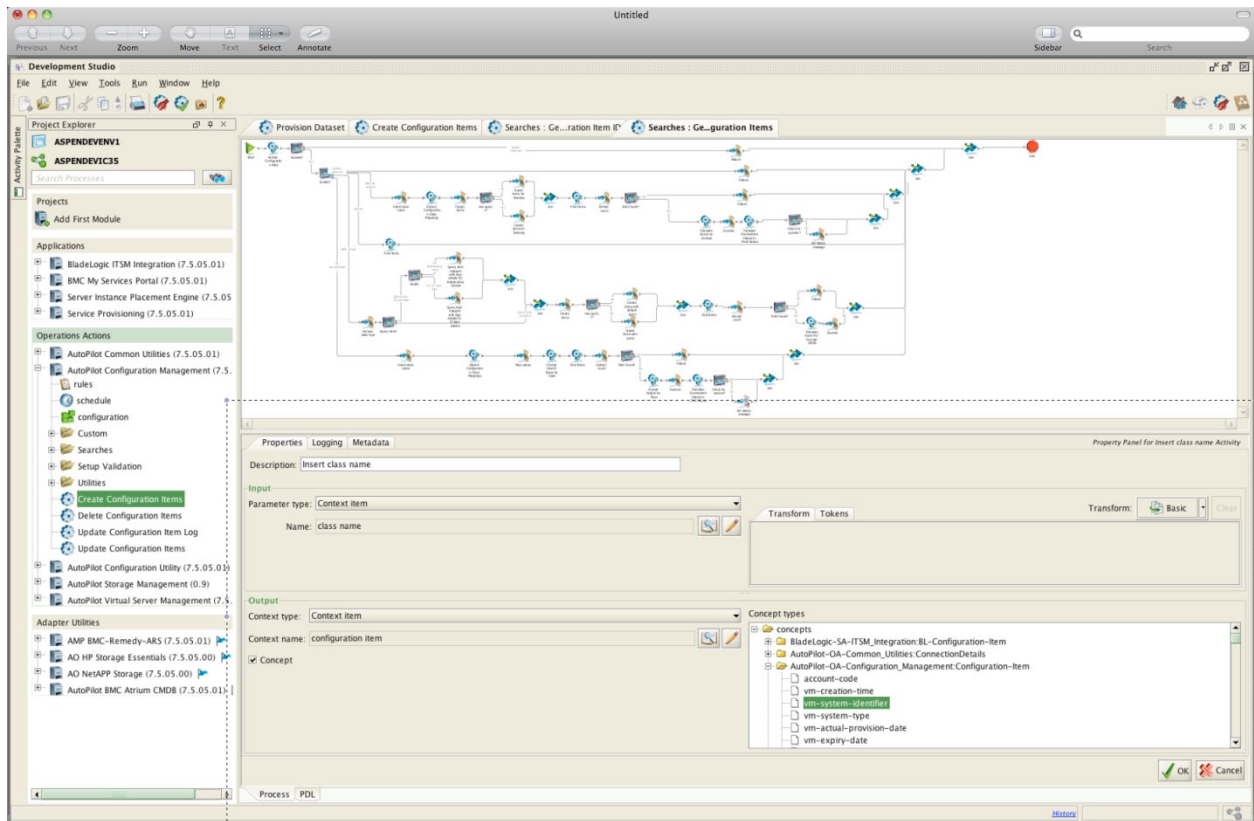


Figure 2. BMC Atrium Orchestrator DevStudio: OA Module Configuration and Concept Management

Solution configuration data

BMC Atrium Orchestrator is usually used as the orchestrator that binds multiple end systems into a solution. Such a solution will usually be built using Adapters, OA Modules, and SA runbooks. In this type of solution a Solution Configuration can be defined in the Project Configuration level.

This module should only contain configuration data, and should not contain any processes. It is reserved for configuration data that are specific to any given deployment (e.g. Dev, Test, Prod). Data should be organized in a hierarchy matching the module definitions, where appropriate. It is intended that this module would be deployed to any given grid only once, and that its contents would then be manually maintained via Grid Manager as part of ongoing maintenance and code releases.

All other configuration data should be located in the configuration section of the modules to which it belongs. The contents should be generic for all instances where this module may be deployed and should never be manually edited via Grid Manager except as part of testing.

Any configuration data that are deployment-specific should be located in the top-level Configuration module described above.

Module folders

Where possible, processes within a given module should be located in directory structures that provide logical grouping of function or technology. Naming of these directories will be based on the specific functionality or technology. However, certain names should be consistent across all modules.

These are as follows:

- » **Utilities** — Utilities are processes of a generic nature used by higher level processes. These should not be called from outside of the module in which it is defined.

- » **Setup validation** — These are processes provided for testing or validation of the module after installation. These should not be called from any other processes (except perhaps other validation processes). Setup validation is a very important part of a module. It demonstrates the usability of the modules and the required parameters and formatting. This is doubly important in testing and in the case that workflows are outsourced. It is also a vital part of Unit Testing.

Module definitions

All workflows developed for a solution should be located in the applications section. The *Operations Actions* and *Adapter Utilities* sections should be reserved for BMC-supplied code or highly reusable workflows used to interface with external applications or to perform some highly specific function.

Projects section

Modules defined in the projects section should relate to high-level business processes and should be grouped based on business function. They should only contain references to solution environment variables used by processes defined at the applications layer. They should not contain references to lower level processes.

Application section

Modules defined in the applications section are referred to as runbooks, and implement the business process. They should contain processes relevant only to the solution being created. Modules and processes defined in this section will form the basis of business processes defined in the projects section. An example is Continuous Compliance for Servers.

Process coding conventions

1. One process for one operation makes for reusable processes.
2. Clearly defined and minimum number of inputs and outputs.
3. Minimal switching. If you're doing more than three switches in a process, for example, simplify. Go deep instead of wide.

Keep it simple

Processes should restrict their operations to simple activities. If they start to become too complex, then consider breaking the task down into smaller chunks.

Don't use transformations on process calls

When calling another process, try to avoid performing transformations as part of Context Item mapping. If transformations are required, they should be done as separate assignment operations prior to the process call. This makes the code flow more obvious.

Small single steps

Despite the temptation to perform highly complex data manipulation in XSLT, it will make for more maintainable code to perform the task with a number of simpler transformations instead.

Clear switching

When using switch or loop operators, make sure that you label the operation clearly and also make sure that the label applied to the start is copied exactly to the end. This makes following nested statements easier.

Defaults

Create sane defaults for workflows, but do not assume. If you're defaulting a missing parameter, make sure it's clearly logged.

Logging

Logging aids in debugging and software maintenance. The lessons learned on this topic is log everything everywhere

- » Log on assign/transforms/process runs
- » Log within the processes and beyond
- » Log only the relevant values for the process in question (i.e. don't log the full concepts everywhere)
- » Log the process name in the log message too
- » Log the status and the status messages on ending

Do not be afraid to log excessively, as it can be turned off during production in the Grid Manager. The performance overhead for logging is minimal and the human time-savings is substantial. In distributed development environments, excessive logging will help the various developers and the subsequent maintainers.

If there are any unknowns for a process, if the process is not finished for whatever reason, if you found a bug, or if you have any clues that you think might help the next person, please (PLEASE!) add a log label such as:

- » TODO
- » TESTME
- » HARDCODING xyz here
- » DANGER
- » BUG

This will be seen as a note to the next person working. Sometimes there simply is no time to fix everything. If you encounter such a thing in someone else's workflow, please make a note of it as well, and document it somewhere obvious. This holds true for out-of-the-box workflows as well as the custom-made ones.

Compensation

Compensation is a BPEL concept intended to describe what steps to take when things break in an unexpected, unrecoverable way.

For BMC Atrium Orchestrator, compensation should only happen when something at the code level breaks. This is most often related to an adapter not being able to execute a command or a required parameter with no value (the root cause can be verified by examining \$BMC Atrium Orchestrator_HOME/[peer type]/logs/grid.log).

Note that searching a database and retrieving no records, executing something on the command line that fails, or looking for a CI that doesn't exist in a CMDB are not causes for compensation.

These would be considered "expected" failures that are handled as a regular part of workflow (usually with a switch Activity).

DEBUGGING

The most effective use of the debugger is to set a breakpoint on an Activity shortly before the step in the process that is of interest. Context item values can be viewed and changed during the Process execution.

Logging should be added whenever there are context items with values of interest. There is no significant performance hit for having a lot of logging.

Further, logging can be disabled at the grid level via Grid Manager. This enables frequent logging throughout the development and testing phases.

Logging is then turned back down during production. Turning the logging level back up in production can provide rapid debugging.

While adding logging should be a standard activity when developing workflow, occasionally it is somewhat time consuming and inefficient to wait for a process to complete before sifting through the logging statements to figure out where something went wrong. The debugger excels in these situations.

METRICS

The Metrics Activity allows process developers to capture any metrics that are deemed relevant at any point within a process. These will be recorded to the same metrics database as process metrics that are turned on or off in the process properties pane (opened by double-clicking in the background of the process window).

Note that process metrics need not be enabled for the metrics activity to work. Process metrics are not recorded when testing processes within DevStudio.

TIPS

1. Lowercase everything before making any comparisons if the data is case-insensitive.
2. Escape anything that needs to be escaped. Take into consideration inputs that might take quotes as a part of the value passed.
3. If the number of inputs to a process is more than five, then consider taking the inputs in the form of an xml in order to reduce the number of total inputs.
4. Log everything that is important. Logging has no processing overhead, so more logging is better than less.
5. Do not log requests containing credentials.
6. Avoid creating large and complex processes. Break them down into simpler chunks.
7. Do not make more than five assignments in a single assign activity item.
8. Do not do transformations on process calls.
9. Ensure that every process has an output called status. The format of the same should be as follows:

For Pass:

```
<status>
  <success>true</success>
</status>
```

For Failure:

```
<status>
  <success>false</success>
  <message>Reason for failure</message>
</status>
```

CONCLUSION

The BMC Atrium Orchestrator Platform and the workflows it hosts provide a highly customizable yet auditable environment for deploying complex IT solutions. Today's data centers must support many mission-critical business applications composed of multiple distributed heterogeneous components. Workflow-based automation gives the data center manager the means to quickly create and deploy end-to-end solutions to meet the needs of the business.

The workflow-based solutions, though flexible, allowing rapid development and deployment, can cause untold damage if strict rules of development are not followed. Experience dictates *iterative development*. This has worked for complex solutions developed on applications with distributed heterogeneous components. Control of the requirements, change, and QA processes should not be ignored. These processes work for all other programming languages and methodologies and they work here.

Finally, take the time to develop good application interfaces. These will become critical as the workflows evolve and new features are added and deprecated features are removed. Remember, the BMC Atrium Orchestrator Platform provides a layered arrangement that lends itself to best-practice coding, so please use it.

APPENDIX A

AUTOMATION OVERVIEW: WHAT DOES AUTOMATION MEAN?

Automation means a number of different things, depending on the audience and the topic. Automation can be as simple as using a tool to push a text file from one machine to another; it can also be much more complex, doing that file push as one of a number of steps:

1. Receive an alert about a device abnormality
2. Add logging to the alert to indicate an automated attempt at remediation
3. Suppress the alert to prevent an operator from working on the same problem
4. Perform triage on the affected machine, including running a compliance check against a golden standard
5. Create a change request to update the file found to be out of compliance
6. Create an incident indicating the out of compliance condition
7. Link the change and incident for easy tracking; add change and incident information to the event
8. Upon event approval, push the file to the affected machine
9. Rerun the compliance check, as well as any other post-modification steps, e.g. restarting services, fixing permissions, etc.
10. Close the change request, indicating the steps taken
11. Close the incident, indicating the steps taken
12. Close the alert, indicating the steps taken

Automation can be a function provided by a single piece of software, or the orchestration of a number of tools to accomplish a larger end result.

There are two broad categories of automation: full and semi-automation. There are benefits to both approaches, and a full solution might well include both.

	Full automation	Semi-automation
Definition	Hands-off, behind the scenes, requiring minimal operator intervention	Interactive, visible, involving an operator for approvals, next steps, decision points
Best for	Repetitive processes that do not require making judgment calls, though these can typically be handled by breaking the workflow into steps with user input coming from ticketing systems, change management systems, etc.	Processes requiring judgment calls or a high level of real-time monitoring.
Design implications	Designing from the beginning for full automation will place an emphasis on creating a fully fault-tolerant, scalable solution. Such a tool should be reliable and able to recover from a variety of execution issues.	An initial focus on semi-automation will focus on the operator's user experience, placing emphasis on UI aesthetics and interactive ease of use. Interactive workflows are typically slower to execute, and, as they have someone monitoring them, need not necessarily recover from a failure on their own. As is often the case where UI is considered first, the solidity of the rest of the architecture might be lacking.

Business Runs on IT.**IT Runs on BMC Software.**

Business thrives when IT runs smarter, faster and stronger. That's why the most demanding IT organizations in the world rely on BMC Software across distributed, mainframe, virtual and cloud environments. Recognized as the leader in Business Service Management, BMC offers a comprehensive approach and unified platform that helps IT organizations cut cost, reduce risk and drive business profit. For the four fiscal quarters ended December 31, 2010, BMC revenue was approximately \$2 billion. Visit www.bmc.com for more information.

