Asset lifecycle management for service-oriented architectures

Grant Larsen
Jack Wilber

October 15, 2005

from The Rational Edge: This article examines the use of asset lifecycle management practices, tools, and standards in the development of service-oriented architecture (SOA) solutions. Effective lifecycle management of services enables organizations to apply tools and methods for governing, administrating, applying, and reusing these assets to augment the advantages of SOA development.

Service Oriented Architectures (SOAs) hold great potential for organizations seeking to become more responsive to rapidly changing market demands and business priorities. With an SOA, architects can combine and reuse services -- individual business functions and processes typically realized as services -- to assemble complex business solutions.

From a business perspective, SOAs improve responsiveness and shorten time-to-market, because they allow organizations to quickly assemble solutions from business-aligned building blocks. From an IT perspective, SOAs can significantly reduce the complexity of large-scale development efforts, enable greater flexibility, and accelerate the delivery of solutions that deliver real value to the business by automating and integrating business processes.

Underlying all of these benefits is the concept of reusability. Services are assets that grow in value as they are reused. The less time and effort you need to apply a service to a new solution, the greater its value. Said another way, the value of the service is preserved as it is reused more and more, and as the level of effort to reuse it is minimized. As a result, effective lifecycle management for services is vital to the success of current and future development initiatives. It enables organizations to apply tools and methods for governing, administrating, and applying these assets in a way that amplifies the benefits of SOAs.
When development teams can draw from a pool of certified, proven assets and solutions, organizations can improve responsiveness, time-to-market, and productivity. They can also mitigate complexity, because developers can leverage those assets without needing to understand their inner workings.

What is an asset?

About five years ago, a consortium of software industry leaders -- including IBM, Rational Software (before its acquisition by IBM), and Microsoft -- began exploring ways to help organizations re-purpose software investments. This consortium defined an asset as:

... a collection of artifacts that provide a solution to a problem for a given context.

An asset may have variability points that allow users to customize the asset by setting various parameters. An asset that can be treated in this way is called a template. Today, IBM Rational tooling is implemented with this definition in mind.

Assets also include instructions or rules for their use in order to minimize the time developers need to discover, analyze, consume, and test the asset. Assets also describe the development and business context in which the asset can and should be reused.

The kinds of artifacts the asset contains depend on the reuse context. For a development context, the asset may contain requirements, models, source code, and tests. Those who build services should include these kinds of artifacts, to help other developers reuse the service efficiently.

An asset might be a service, pattern, component, or other solution element. When building a service, developers commonly reuse patterns, components, or legacy system artifacts. Then, along with other services, they can assemble it into a composite application as part of an SOA solution.

Note that the definition of an asset is strikingly similar to that of a pattern. This is by design, as the fundamental concepts behind both pattern and asset pull together context, problem, and solution. The difference is in the details. An asset is a more general concept than a pattern. Strictly speaking, the variability points of an asset are at the artifact level, whereas a pattern will have parameters and participants (i.e., kinds of variability points) that apply to the entire pattern, and not necessarily to any particular artifact.

RAS: A standard for assets

After the consortium had answered the question, What is an asset?, new questions arose: How do we organize and structure an asset? What information do we need to know about it? They created the Reusable Asset Specification (RAS) to answer these questions and then submitted it to the OMG (Object Management Group), which adopted it as a standard in 2005.

As noted above, there are many types of software development artifacts, and these may exist in many forms and reflect a variety of styles. This diversity can increase the costs of discovering, comprehending, and reusing other authors' artifacts. By specifying a way to organize, structure,
describe, and package these artifacts, RAS provides consistency and predictability across assets, which significantly reduces the costs of asset management and consumption.

To illustrate the value of standardizing asset packaging, consider how standardization has helped overnight shipping companies. By providing standard envelopes and boxes to customers and requiring consistent information on the shipping label, these companies were able to build sophisticated tracking systems. In addition, these policies facilitated better planning and decision-making across the business, from determining the width of conveyor belts, to estimating how many packages can fit on a plane, and how many flights are required.

Asset standardization promotes efficiency in a similar way, enabling greater efficiency and automation in software development tools and improving productivity for the teams that use them. Using the RAS, developers can package each asset in a meta-data wrapper that includes top-level attributes, such as the asset's name, version, and description (see Figure 1).

![Figure 1: The RAS meta-data structure](image)

As Figure 1 shows, a RAS asset has a *classification* section that facilitates searching and browsing; this section might include simple name/value pair descriptors, and a declaration of contexts, such as a specific domain, development, or deployment context.

The *solution* section shown in Figure 1 is the "meat" of the asset; it describes the collection of artifacts that provide a solution.

The *usage* section provides guidance on applying and customizing the asset, using variability points. Some usage information may be automated using scripts and wizards, which are stored in the solution section with the other asset artifacts.

The *related assets* section defines the asset's relationships to other assets and helps to create collections or families of assets to form larger-grained solutions.
To support varying degrees of reuse, formality, and process maturity in organizations, many of the RAS asset sections are optional. RAS can also be extended and customized via a profile. The OMG currently offers three profiles: the Default profile, the Default Component profile, and the Default Web Service profile.

The Default profile is used to package any kind of software asset, while the other two profiles are intended for use with components and Web services, respectively.

**The asset lifecycle**

After gaining consensus on what defines an asset and how to standardize its organization and packaging structure, Rational’s next step was to describe more fully an asset’s lifecycle: the processes by which it is identified and discovered; harvested and created; certified and published; reused and measured; and ultimately retired. As Figure 2 shows, the asset lifecycle includes four distinct workflows: identification, production, management, and consumption.

![Asset program management](image)

**Figure 2: The asset lifecycle and its workflows**

IBM advocates an approach to software development called **asset-based development (ABD)**, an integrated solution grounded in these workflows, which supports each workflow with process, tooling, and standards. Using process guidance provided by IBM Rational Unified Process®, or RUP®, teams know who is expected to do what tasks and when. They also know how to use products in the IBM Rational Software Development Platform to develop architected solutions, models, and other artifacts based on a set of well-defined standards, including RAS and UML. IBM also offers several RUP plug-ins, including an ABD plug-in and a RUP plug-in for SOA that integrates support for service-oriented architecture and service-oriented solutions into the RUP framework, with SOA-specific concepts, guidelines, activities, artifacts, and tool mentors.
The asset lifecycle, the project lifecycle, and RUP

It is important to note that the asset lifecycle is independent from, yet connected to, the project lifecycle. Assets are produced as part of the asset lifecycle, and they may be consumed in many projects, each with its own lifecycle. Later, independent of any project, the asset may be retired, and thus end its participation in the asset lifecycle. Assets typically grow in maturity: for example, from being reusable for a single team to a larger reuse context that spans multiple teams and projects.

In much the same way, the roles and use-case activities associated with ABD and the asset lifecycle are independent from, but related to, roles and activities of RUP. The kinds of assets individuals are interested in -- as well as the tools they use to access and manipulate those assets -- are generally dictated by the ABD and RUP roles they are fulfilling.

For example, an individual acting in the role of architect in RUP may also assume the role of asset consumer in the asset lifecycle. In this case, the architect will search for, browse, and reuse assets. The architect will likely look for architecture-related assets, such as models, and will likely use a modeling tool, such as IBM Rational Software Architect.

Similarly, specific combinations of asset lifecycle use-case activities and RUP activities can correlate with certain RUP phases. The architect, acting as an asset consumer, will typically search for assets during the Inception and Elaboration phases of RUP. However, a tester, also acting as an asset consumer, would likely search for test cases or other assets during the RUP Construction and Transition phases. There are no constraints on when individuals in these roles may search for assets, but for any particular RUP phase we will see more searching done by one role than another.

In short, the roles and activities of the asset lifecycle map naturally to existing development processes, including those described in RUP. Further, each specific combination of roles and activities suggests what kinds of assets will be of interest and what tools will be most helpful.

The asset lifecycle is exercised within the confines of a project, which is where we see most of RUP applied. The asset lifecycle is also exercised beyond the boundaries of any given project; as can be seen by the Asset Management and Asset Production boxes at the bottom of Figure 3, which illustrates where Asset Identification, Asset Production, and Asset Consumption can occur within a project.
Figure 3: The asset lifecycle is exercised both within the confines of a RUP-based project, as shown by the Asset Identification, Asset Production, and Asset Consumption blue boxes, as well as beyond the boundaries of any given project, as can be seen by the Asset Management and Asset Production blue boxes at the bottom of this illustration.

We can also consider these relationships from the perspective of the asset lifecycle itself. Figure 4 maps the various ABD workflows across several projects. In this example, we have articulated some high-level activities that occur within respective ABD workflows. The activities for Asset Identification are performed on Project A, whereas the Asset Production and Management activities are performed by the Asset Manufacturing team, not aligned with any particular project. And finally the Asset Consumption activities are performed on Project B.
Figure 4: Mapping the various ABD workflows across several projects

An application development project may conduct Asset Production; however, the intended reuse scope for such assets should be constrained until the asset can be evaluated in broader reuse scopes.

To summarize, the ABD workflows can be mapped on to the RUP phases as shown in Figure 3, and are applicable across project boundaries, as shown in Figure 4.

Tools to support the asset lifecycle

Although defining a standard for packaging and describing assets and a set of asset lifecycle workflows were important milestones, many development teams still needed the tools to support RAS and the lifecycle workflows. As Figure 5 shows, the IBM Rational Software Development Platform fills this need, leveraging RAS to realize and implement the asset lifecycle workflows.
Figure 5: IBM Rational Software Development Platform tools that support RAS and asset lifecycle workflows

Using these tools, teams can create, certify, and package services as reusable assets, expressing their Web service interfaces in WSDL (Web Service Definition Language) and having them point to a UDDI (Universal Description, Discovery, and Integration) registry. Then, by storing the services and related assets in a RAS repository (more about this later), team members can efficiently search for, measure, evaluate, and -- most important -- reuse them.

Roles in asset lifecycle management

Each asset lifecycle workflows has an associated set of use cases that defines the actors and activities performed in the workflows. RAS provides the necessary meta-data and extensible structure to support these actors (or roles) and use cases throughout the lifecycle. Each actor in Figure 6 has a specific set of responsibilities. Defining roles in this way enables greater flexibility as a single person may take on more than one role, or multiple people may fulfill the responsibilities of a single role.

Figure 6: Roles and responsibilities in the asset lifecycle

As Figure 7 shows, the asset identification and asset production workflows use the top-level, solution, usage, and related-assets sections, whereas the asset management workflow mainly leverages the top-level and classification meta-data, and the asset consumption workflow uses all the meta-data.
Figure 7: Each workflow in the asset lifecycle leverages RAS asset information

Tools in the IBM Software Development Platform support the activities performed in each workflow, as we shall discuss below.

IBM support for the asset identification workflow

The first step in the asset identification workflow is to identify the problem and a potential solution. The person acting as the asset identifier (see Figure 8) will then search for an applicable candidate asset and artifacts, and submit that candidate service by packaging it as a RAS asset and storing it in a RAS repository. The goal of this individual is to identify all potentially useful existing artifacts and to lay the groundwork for the organization to produce and consume the service.

Figure 8: Asset identification use cases

IBM WebSphere Studio Asset Analyzer helps identify existing services for reuse. Developers, analysts, and operations teams use WebSphere Studio Asset Analyzer to better understand
enterprise software assets, and to enhance, maintain, and reuse them more efficiently. WebSphere Studio Asset Analyzer provides insight into assets and their relationships through interactive textual and graphical reports. And, it identifies business process, flows, and solution dependencies at multiple levels to help drive the asset identification workflow.

**IBM support for the asset production workflow**

The individual or team serving as the *asset producer* will transform the candidate asset into a truly reusable asset, either by developing additional artifacts, refining existing artifacts, or both. In this context, "producing" a service involves modeling, coding, and constructing it, using a tool such as IBM Rational Application Developer, and then positioning the service to be consumed and reused later in the asset lifecycle (on any number of projects), using tools such as IBM Rational Software Architect and IBM Rational Software Modeler (see Figure 9).

![Figure 9: Asset production use cases](image)

**Variability** refers to how customizable the asset should be. Some assets can be consumed as is, without any modification or customization. Other assets can -- and in some cases must -- be adapted by the consumer before they are reused.

**Articulation** refers to the degree of completeness for the solution the artifacts provide. If the artifacts specify, but do not implement, a solution, then the asset has low articulation. In contrast, if the artifacts specify and implement a solution including supporting files, models, requirements, tests, and so on, then the asset has high articulation.

**Granularity** refers to the size of the asset with respect to the functionality it provides and the artifacts it contains. The asset producer must decide whether to place all the artifacts in a single asset, or to break them out into many smaller, but simpler, assets. If the lifecycles of the associated artifacts vary widely, it is a good idea to use a finer granularity. A service may also be
decomposed into multiple assets, each of which exposes a different interface -- SCDL (Service Component Definition Language) or WSDL (Web Service Definition Language), for example -- with usage scenarios that describe that particular interface.

Figure 10 illustrates a service (called Service X) packaged with a collection of artifacts and supporting meta-data. This asset can be stored in a RAS repository, searched, browsed, and ultimately reused for other application development needs.

![Service X at Development Time](image)

**Figure 10: A RAS asset with artifacts and meta-data**

Good packaging -- like beauty -- is often in the eye of the beholder. Although packaging principles may seem informal, there are some fundamental principles to follow. First, the user's needs should guide the packaging effort. Second, the packager should evaluate the lifecycle of each of the asset's individual artifacts.

Figure 11 shows a service packaged with multiple assets. Note that the interfaces are packaged into their own respective assets, because the lifecycle of artifacts in the main entry point asset (requirements, models, tests, etc.) are significantly different from the lifecycle of the interface assets. Leveraging the related-asset section in RAS, the asset producer creates associations to the interface assets for the service.
IBM Rational Software Architect and IBM Rational Software Modeler both incorporate capabilities to automate the process of packaging and producing assets. Using the RAS export wizard, developers enter key RAS meta-data, select the artifacts for the asset, and then export the RAS archive file (.ras file) to a target location, such as the RAS Repository for Workgroups.

**IBM support for the asset management workflow**

The asset management workflow focuses primarily on the governance of services. Although governance is required across asset development, deployment, and runtime, we will focus only on development.

Effective asset management depends heavily on establishing a well-defined governance process that describes roles, policies, states, and transitions. This process should detail how to retire assets, track asset metrics, create a classification schema, certify and review assets, and publish reusable assets (see Figure 12). It can also describe how to create custom RAS profiles.
Figure 12: Asset management use cases

Development-time governance includes activities that cross all asset-based development workflows, though we often think about it solely in asset management. Figure 13 outlines sample development-time governance activities for each workflow.
Figure 13: Development-time governance activities across the asset lifecycle

To help teams establish effective governance practices, the IBM Rational supplies RAS Repository for Workgroups (see Figure 5), a lightweight repository available on the IBM alphaWorks Website. Used together with IBM Rational ClearCase, this repository facilitates asset management, and provides the following capabilities via its Web service interface:

- Searching and browsing of assets in the repository, using the RAS 1.0 standard repository service interface as well as a second, enhanced interface that supports more complex queries.
- Retrieving asset information in multiple formats, including views of the documentation and feedback, and of the complete asset.
- Publishing assets to the repository, creating and organizing the logical view of assets in the repository, and tracking metrics.

Asset managers can use the RAS Repository for Workgroups to track metrics, such as how many times an asset is searched for, how often its documentation is browsed, and how many times it is imported and reused. Developers and other asset consumers can also use it to rate assets or services and provide textual feedback.

The processes required to review and certify an asset can be automated and enforced with IBM Rational ClearQuest, which walks teams through a series of well-defined steps to certify a service and ready it for publishing. In addition, IBM Rational ClearQuest and IBM Rational ClearCase provide software configuration management capabilities that are central to modifying artifacts and assets.

IBM support for the asset consumption workflow

As Figure 14 shows, the first step in asset consumption is locating the service, either through an explicit search using keywords from the RAS meta-data, or by browsing assets in a RAS repository. The consumer of the asset can then apply the service and provide feedback via the repository.
Figure 14: Asset consumption use cases

IBM Rational Software Architect and IBM Rational Software Modeler include an Asset Explorer that enables teams to easily locate assets in any RAS repository and provide feedback. Rational ClearQuest is also used to provide feedback in the form of defect reports and change requests.

IBM Rational tools also make it easy to apply assets. Integration developers can drag and drop the services' WSDL, which was imported from a RAS repository (via IBM Rational Software Architect), into a BPEL diagram in IBM WebSphere Integration Developer. Figure 15 shows a service in a RAS repository before it is imported.
Figure 15: Importing a service from a RAS repository

After you import the service, you can bring the WSDL into IBM WebSphere Integration Developer by dragging and dropping. From there, you can add the service as a reference partner and include it in BPEL models (see Figure 16).

Figure 16: Accessing a newly imported service

Aligning IT with business goals: Business-Driven Development

To deliver real value to the business, IT departments must align their own goals with business goals and priorities, which should permeate the entire asset lifecycle and inform the activities in each workflow. This is what IBM refers to as Business-Driven Development (BDD). In BDD of service-oriented solutions, business goals and needs drive the entire development process, from design to construction, integration, and testing, as shown in Figure 17.
Figure 17: Business-Driven Development allows an organization to trace business goals throughout the development process for service-oriented solutions.

BDD brings together the techniques of Model Driven Development (MDD) and ABD as a key governance aspect. Our approach here is to use these techniques for creating service-oriented architectures (SOAs). SOA provides the primary solution architecture for applying MDD and ABD, which describes the nature and types of assets involved.

BDD begins with a solid understanding of the business and business processes. These processes are modeled and mapped to services. Individual services are constructed, tested, deployed, and ultimately assembled into composite applications using an iterative approach.

Business goals and needs drive the lifecycle: from the business process to the service specification, to service development, construction and composition, and certification and deployment.

We should note that patterns -- a special kind of asset that describes solutions to recurring problems within a specific context -- provide an effective way to consistently align IT initiatives with business goals. Applying patterns for the development and reuse of services is also a powerful technique for boosting productivity and reducing complexity.²
Conclusion

Reusability lies at the heart of asset lifecycle management and service-oriented architecture. Teams that leverage IBM tools, guidance, and resources to manage the lifecycle of their assets have a significant advantage in their SOA development initiatives.

No matter how far along your organization is in its adoption of SOA, it pays to think critically about reusability and the asset lifecycle. Improvements in these areas can help improve productivity, reduce complexity, accelerate development, and ensure better alignment with business goals.

Notes

1 alphaWorks provides direct access to IBM's emerging technology. RAS Repository for Workgroups is available as a free download at http://www.alphaworks.ibm.com/tech/rasr4w

2 A thorough discussion of patterns is beyond the scope of this article; we hope to explore this topic in a future article.

© Copyright IBM Corporation 2005

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)