Open Services for Lifecycle Collaboration
White Paper
May 2009

The Case for Open Services

John Wiegand, Distinguished Engineer, IBM Rational Software

Contents

The Problem

Teams and organizations concerned with software delivery know that they can be much more effective when the tools that they can be used in combination. They want traceability across resources and accountability across processes, without burdensome manual overhead. However, the tools landscape itself, having emerged organically from point tools aimed at solving specific narrow needs in the software delivery lifecycle, can present challenges for these organizations.

Let's consider a typical organization. They start with a range of tools from multiple vendors, often complemented with internally-developed custom tools. Of course, they want traceability between the resources across the lifecycle like requirements, tasks, source code, and test cases. However, rather than uniformly connected resources, they often find integrations through specific bridges between each pair of tools – brittle connections based on unique tool-to-tool APIs. Moreover, their data is often buried inside the tools. When one tool needs to access another tool's data, a bridge is required, implemented through a vendor-specified API often tied to a specific platform or language. And when a tool needs to record additional information, yet another bridge is required.

This tightly coupled network of custom bridges can be vulnerable to everyday disturbances -- changes like upgrades of the underlying OS or API revisions from the vendors. Additionally, individual tools tend to each have their own vocabularies, providing alternate names and descriptions for comparable concepts (or sometimes: different tools all using the same term with subtly different meaning). Even when tools can share data they may be unable to share meaning, and a single logical asset can be scattered over multiple tools, requiring (even more) custom bridges, translation and synchronization.

Highlights

An ideal solution would provide a uniform architecture and set of protocols that allow resources from loosely coupled tools to be integrated in a consistent way.

We can pattern a loosely coupled integration architecture after the Internet.

Characteristics of a solution

An ideal solution would provide a uniform architecture and set of protocols that allow resources from loosely coupled tools to be integrated in a consistent way. However, if any single vendor were to invent such an architecture, it would just create a bigger black box to which other vendors would need to build bridges. If instead existing open standard technologies were leveraged, the value of the resulting integrated world would outweigh the incremental cost of participation. The Internet has precisely these characteristics – in fact, we can pattern a loosely coupled integration architecture after the Internet[1].

In addition to supporting Internet-style integration, an ideal solution would have these architectural characteristics of the Internet:

- **Scalable** - supporting unlimited numbers of users and resources.
- **Distributed** - supporting globally dispersed users and resources.
- **Reliable** - working well over a wide range of connectivity profiles.
- **Extensible** - an open-ended set of resources with extensible representations and protocols/services for operating on them
- **Simple** - easy and flexible for tool and content authors to learn and use, and that does not depend on close cooperation or continuous coordination between vendors.
- **Equitable** - equally available to all participants, from individual projects to large vendors; open-source, in-house or commercial development; with no barriers to participation.

Practical steps

Transitioning from the ideal to the practical -- how can we leverage the Internet architecture to achieve our goals for improving lifecycle collaboration and for sharing lifecycle resources? We propose three incremental steps that a tool writer can adopt. These steps transform lifecycle resources

[1] W3C: *Architecture of the World Wide Web, Volume One* , http://www.w3.org/TR/webarch

Highlights

Open Services for Lifecycle Collaboration is the moniker for this vendor independent approach to lifecycle integration.

into "hyper-data," just like hypertext enables fully connected, flexible content (Tim Berners-Lee's discussion on Linked Data[2] explains this concept well). Open Services for Lifecycle Collaboration is the moniker for this vendor independent approach to lifecycle integration. In each step, we articulate an Internet standard mechanism that can be used in a uniform way.

Step 1: Internet URLs for resources

The first step is to provide a universal address for each resource – whether it is a requirement, a test case, a defect, or anything else. The web mechanism for defining a global address is a Uniform Resource Locator (URL). Like URLs as web page addresses, we're using URLs to provide an address for each of our resources. Once a resource is URL-addressable, it can be referenced from any web page, tool, or other lifecycle resource.

Once a lifecycle resource is URL-addressable, it can be referenced from any web page, tool, or other lifecycle resource.

Step 2: Shared resource formats

Although the first step provides an addressing scheme for each resource, it doesn't inform a tool about what's "inside" the resource – the content is still unknown. Although on the surface this sounds like a limitation, it's actually a design characteristic of this flexible architecture – resources aren't restricted to a fixed set of pre-defined types. For example, a requirement could be represented by a text document describing the requirement, an image showing the requirement, or an XML document, defining the attributes of

[2] W3C: *Design Issues for the World Wide Web – Linked Data*, May 2007, Tim Berners-Lee, http://www.w3.org/DesignIssues/LinkedData.html

Highlights

We can do more with the resources when we know some details about the format of the resource.

Common elements enable tools to create resources and to view and translate them into a local format.

the requirement. A tool that tracks, for example, the relationship between a test-case and a requirement need not understand the requirement's contents; it need only know of its existence and location. When its users want to see the contents of the requirement, it functions like a browser, getting the resource and handing it to an appropriate tool.

Although we value this flexibility, we can do more with the resources when we know some details about the format of the resource. Therefore for our second step, we suggest that lifecycle resources be defined in XML and use common elements. Now, the resource transitions from being a black-box to semi-transparent; any tool can examine the common elements of these resources. For example, a tool could view the description of any lifecycle resource (assuming a common description element was defined). In addition to the common elements, a tool can augment the resource with additional elements to record tool-specific information.

One valuable attribute of this Internet-like approach is that tools can share resources without becoming tightly coupled. Tools only need to agree on common elements; they can change the format, content or meaning of their own private elements at will. The architecture can even accommodate lack of complete agreement on the common elements: the nature of XML allows a tool to quietly ignore elements it has no use for or doesn't understand. This Internet characteristic is sometimes called "graceful degradation", in contrast to the typical tool behavior of catastrophic failure when faced with less-than-total compliance.

Common elements enable tools to create resources and to view and translate them into a local format, but more resource design guidance is required to enable deeper collaboration – for example, for a tool to modify a resource that it has not created. This is important to allow tools that share common resources to record additional information in existing resources and to establish references to other resources or provide other tool-specific information. For example, a business analyst working in a traditional requirements management tool might document some requirements as

Highlights

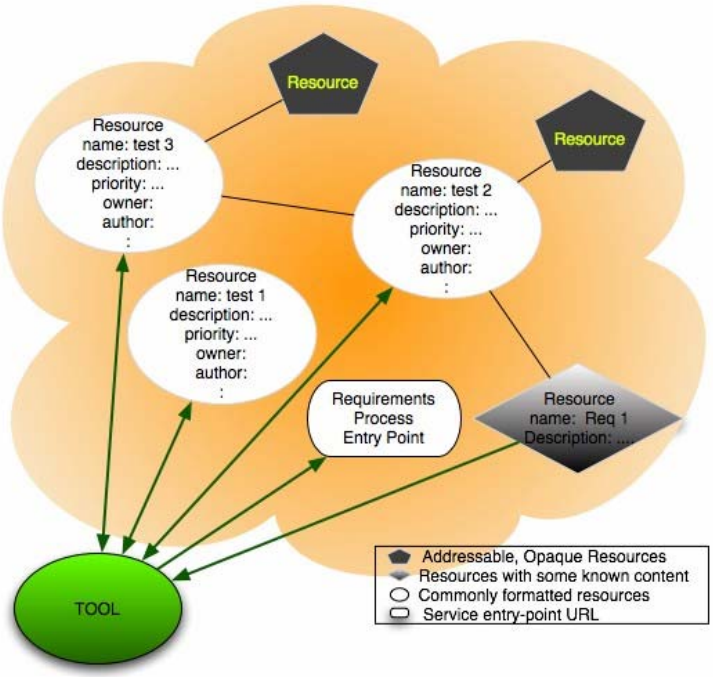
Once we have shared resource formats, we need to provide an appropriate service interface to them. Following the theme of Internet-inspired simplicity, we have adopted RESTful web services as our programming model.

Tools need not be coded to or adopt language or platform-specific API's, enabling much looser coupling between tools.

Use Cases. A designer working in a UML modeling tool might then augment them, without the necessity for tight dependencies on tool versions or a formal cutover and conversion from "requirements phase" to "design phase" typical of today's integrations.

Step 3: Shared resource services

Once we have shared resource formats, we need to provide an appropriate service interface to them. Following the theme of Internet-inspired simplicity, we have adopted RESTful web services as our programming model. For example, the designer, armed with his UML tool, might observe the need for new requirements to be documented as use cases. He'd like to create those requirements and have them participate in the requirements management tool's approval process. With a RESTful architecture, that tool might only need to know the URL to which new requirements are POSTed to trigger this. The UML tool need not be coded to or adopt the requirement management system's language or platform-specific API, enabling much looser coupling between the two tools. Additional services can be built on this base, offering the possibility for common query and reporting, traceability analysis, and process support across the artifacts.



Highlights

Notice that each of the steps enumerated above can be applied to existing tools as well as new tools.

Learn more at http://open-services.net.

Enabling possibilities

Each of these three steps enables improved integration possibilities. Individual tools may choose to provide additional and valuable custom steps of integration. The scope of Open Services for Lifecycle Collaboration is purposefully limited to provide some basic building blocks that can benefit software delivery organizations and tool vendors alike, not to replace or constrain the entire universe of lifecycle tool integrations.

Notice that each of the steps enumerated above can be applied to existing tools as well as new tools. One of our design points is to choose integration mechanisms that can enhance existing resources, analogous to the way that Web Services can be used to “wrap” existing services regardless of implementation.

What IBM is doing

IBM arrived at this Internet-inspired approach to lifecycle collaboration from our experiences with our own customers’ challenges and from our extensive work on the Jazz platform. We're sharing our thinking with the community -- describing the approach that has emerged from our efforts as well as our experiences in resource design. To begin with, we outlined some typical lifecycle resources and their relationships, and suggested a few sample resource descriptions. Now we are applying this approach to our new product development to enable business analysts, developers, and testing organizations to collaborate across a set of tools. Our Collaborative ALM initiative (http://jazz.net/projects/collaborative-alm) is using this approach to integration.

What you can do

If you are creating tools or tool integrations, we invite you to dig further into the details of our three-step approach to lifecycle collaboration and to consider this integration architecture for your tools. Ask yourself the

following questions:

- What are your resources, and their formats?
- Are you interested in collaborating on defining shared formats and services?
- Are there other steps of integration we should consider?

If you are an organization involved in software delivery, we hope you find this approach appealing. In the end, our goal is to allow tools to readily share lifecycle resources, enabling you to more easily integrate, manage, and evolve lifecycle tools and processes for software delivery in response to new business demands. Encourage your tool providers to support this integration architecture so that we can together eliminate some of the challenges and unnecessary barriers to integrated software delivery.

References

- Open Services for Lifecycle Collaboration (http://open-services.net)
- IBM's Jazz initiatives (http://jazz.net)

Acknowledgments

Thanks to my colleagues who reviewed and provided helpful input to this paper: Steve Abrams, Scott Bosworth, James Branigan, Jim des Rivieres, Robyn Gold, John Kellerman, Simon Johnston, Martin Nally, Scott Rich, and Carl Zetie.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
05-09