Rational. software

The business value of open collaboration

How interoperability is fostering a new ecosystem for manufacturers and consumers in the software delivery arena

Carl Zetie
Strategist, IBM Rational

Scott Bosworth
Program Manager, IBM Rational

Contents

2 *Introduction*
3 *It takes a village*
5 *The greatest integration on Earth*
6 *OSLC: An open forum for collaboration*
7 *The business value of open markets*
11 *The openness of OSLC*
13 *The OSLC community*
14 *Conclusion*
15 *Appendix A: The technical architecture of OSLC*
17 *Appendix B: Creative Commons Licenses and Copyrights*

Introduction

Software is everywhere. It runs our business processes, back-office functions, and customer transactions. It drives our Web presence, online business, and customer relationships. And increasingly, it's embedded in the physical products and the infrastructure of everyday life -- from automobiles, consumer appliances, phones, and personal devices, to power grids, traffic systems, and health care. Everything we use is increasingly made smarter by software and connected through the Internet.

Consequently, the ability to rapidly and reliably enhance enterprise software systems as well as the ability to deliver smarter products and services have become a critical differentiator for businesses that face increasing competitive pressures. To meet these challenges, business leaders need their software delivery process to work at least as well as their other business processes do. They need to have predictable delivery cycles; cost-to-value management; and continuous visibility into work in progress. They need to be able to rely on the delivery of the software elements of business initiatives and products – but all too often, despite the best efforts of vendors and users, they can't.

Even though development and delivery team members are smart, hard-working, and dedicated to satisfying their stakeholders, somehow problems still creep in. Requirements get lost in translation from analysis to construction. Test cases get missed, causing old bugs to reappear. Project progress and status is hard to decipher. Even in the most successful, well-run organizations, too much effort is wasted on mundane, administrative tasks such as data transfers or status updates.

Highlights

Why is it so hard for smart, hard-working, dedicated professionals to successfully deliver software projects? One major barrier lies in the tools that software development professionals depend on. More accurately, it lies between those tools. The software delivery process today is reminiscent of a 1950s office. The simple act of sending a letter might involve the following steps:

- A manager dictates the letter to a secretary, who takes it down in shorthand
- The secretary delivers the shorthand to the typing pool
- A typist transcribes it, making a carbon copy, then sends it back for review
- The manager reads it over and marks any corrections
- The secretary takes it back, and the wasted carbon copy is discarded
- This cycle repeats until the letter is satisfactory
- The manager signs it and the secretary mails it out
- Finally, the secretary files the carbon copy in case it is ever referred to again

The software delivery process today is reminiscent of a 1950s office, where the simple act of writing a letter involves multiple handoffs. Unfortunately, this quaint last-century vignette is all too reminiscent of the process handoffs in modern software development.

Just imagine the number of ways that this process can go wrong: the secretary mishears the dictation; the typist misreads the scribbled shorthand; or the manager realizes he omitted critical information when he reads back his own letter, to name just a few. Each handoff introduces the possibility for errors and omissions, causing costly rework and delays, not to mention the expense of verification required at every stage to make the process go right. Unfortunately, this quaint last-century vignette is all too reminiscent of the process handoffs in modern software development!

In this paper you will learn about the value of "openness" in business computing, including open standards, open source, and open interfaces. We will also describe an initiative that breaks down the barriers between tools so that teams can collaborate seamlessly, communicate effectively, and provide the visibility that the business needs – without requiring companies to retool and reskill their development organizations.

It takes a village

Throughout the history of the software development industry, vendors have created increasingly helpful and sophisticated tools that have dramatically increased the productivity, accuracy, and effectiveness of every role in the software development lifecycle. But often the effect has been as if we replaced those 1950s typewriters with word processors, and left every other element of

Highlights

Individuals have better tools today than they have ever had, but the process itself is still flawed.

the process in place. Individuals have better tools today than they have ever had, but the process itself is still flawed. No matter how good the business analyst's requirements tool is, effort will be wasted if she has to hand off those requirements to designers (as if she were an executive dictating to a shorthand-recording secretary); and it doesn't matter how good the test automation software is if the defects get passed back to development unreliably (as with hand-scribbled notes on a first draft of a letter). Development organizations need a solution to the challenges that face the team across the lifecycle, rather than merely the challenges facing each individual: in other words, what they need is collaboration.

In an attempt to meet this need, the industry has seen the rise of successful vendors offering cross-lifecycle tools, such as those provided under the IBM Rational brand, that provide the necessary integration and interoperability. However, it's rare that a software delivery organization relies solely on a few tools from a single vendor. More often, teams bring to bear a range of commercial, open source, and in-house developed tools, and often employ competing tools in different parts of their organizations. Realistically then, this discussion can't be about a specific tool suite or small number of vendors; moving beyond the 1950s analogy requires an industry-wide perspective.

It's rare that a software delivery organization relies solely on a few tools from a single vendor. More often, teams bring to bear a range of commercial, open source, and in-house developed tools.

The industry has made numerous attempts to solve the burning need for tools interoperability, but each of them foundered for some combination of the following reasons:

- The scope was too ambitious. Any attempt to define a comprehensive "integration map" for the whole software lifecycle was doomed to be obsolete before it could be finished.
- It depended on unrealistic levels of vendor cooperation. Each vendor involved in standardization efforts would fight to entrench its own proprietary advantages in the standard, and each would just as fiercely resist attempts by its competitors to do the same.
- It required "rip and replace" adoption by user companies. Under some proposals, customers would have to replace their existing tools with new ones. At the very least, they would need to migrate data to a single common repository. That was too high a barrier for potential adopters.
- The collaboration was not sufficiently open. Often, the invitation to participate in a standardization effort was driven by competitive motivations rather than even-handed inclusiveness, leading to the omission of key potential contributors.

Highlights

- The focus was on what vendors wanted, not on what their customers needed. As a result, protectionist compromises limited the business value that user companies could obtain.

Lacking any lifecycle-wide strategy for integration, vendors have resorted to the next-best alternative: individual bridges between pairs of tools, using vendors' published proprietary APIs. The disadvantages of this approach are many.

Lacking any lifecycle-wide strategy for integration, vendors have resorted to the next-best alternative: individual bridges between pairs of tools, using vendors' published proprietary APIs. The disadvantages of this approach are many, and include:

- The number of integrations rises exponentially with the number of tools. Developing and maintaining the integrations is expensive for vendors, and operating them is burdensome for customers.
- Coverage is limited. Each vendor tends to integrate with the tools of its allies, and ignore or exclude the tools of its competitors, to the detriment of customers who would simply like to choose the best tool for each job.
- The integrations are brittle. Pairs of tools become tightly dependent on each other's internal structures or detailed behaviors, and upgrading one product can break the integration. And worse, since each of those tools is on its own release schedule, customers using a variety of tools may struggle to find a mix of releases of the interdependent products that work together.

The greatest integration on Earth

The IBM Rational organization took a step back and asked: What kind of integration approach would it take to make lifecycle collaboration successful?

Faced with this pressing business need and historical difficulty, the IBM Rational organization took a step back and asked: What kind of integration approach would it take to make lifecycle collaboration successful? We decided that the architecture of the solution would have to be:

- *Incrementally adoptable by user companies.* It must not require wholesale replacement of existing tools or skills. Instead, it must allow implementation piece by piece, with each step delivering positive value.
- *Technically acceptable to vendor companies.* It must not mandate a specific implementation technology or repository, nor require vendors to make wholesale changes to their existing products or give up proprietary product differentiators.
- *Authentically open.* Participation would have to be open to any vendor, open source project, or in-house implementer, regardless of competitive positions or alliances.

Highlights

We realized that the best precedent is also the largest example of "integration" that the world has ever seen: the World Wide Web itself.

As we looked around for a successful means of integration that fit this profile, we realized that the best precedent is also the largest example of "integration" that the world has ever seen: the World Wide Web itself. By conforming to a modest set of standard interfaces and a common architectural principle, anybody can participate in the Web by creating a Web site or publishing a blog; and any browser can consume any of the billions of pages that make up the Web.[1] Many different types of content coexist, and new types are easily added without disruption. Furthermore, the simple and flexible architecture of the Web, and in particular its common integration mechanism of resource links, allows the emergence of new uses for all that content that did not exist when the Web was first invented, such as search engines or content aggregation.

On the Web, nobody needs permission to create content, nor to consume content, nor to link to content.[2] The result of this open access is that Internet participation has grown exponentially, while membership in older-style online services that restricted content and limited users' choices has rapidly declined.

How could software lifecycle collaboration be more like the open, unrestricted Web? The answer we propose is agreement on basic specifications to enable integration.

At IBM we realized that the way our industry has approached integration in the past is like those closed, limited online services; and we asked ourselves: how could software lifecycle collaboration be more like the open, unrestricted Web? The answer we propose is agreement on basic specifications to enable integration. The vehicle to get there: a forum for cooperation called Open Services for Lifecycle Collaboration (OSLC).

OSLC: An open forum for collaboration

The goal of OSLC is to re-examine how we look at interoperability so that:

- Any tool can be integrated on an equal footing simply by sharing resources and services agreed to in open specifications
- Anyone can participate in the specification process on equal terms, not only vendors but also user companies, industry forums, open source projects, or motivated individuals
- Any organization can freely take advantage of the specifications, whether they choose to contribute to the effort or not
- The mistakes of the past that led to limited coverage and brittle integrations are averted

Highlights

To achieve those ambitions, OSLC embodies three key elements:

- A unifying – and universal -- architectural style that leverages the lessons of the Web.
- A set of technical specifications inspired by real world scenarios, openly published and freely adoptable.
- A transparent and open community process with no unreasonable barriers to participation.

The OSLC organization today is simply a group of people from across the industry who utilize modern on-line collaboration tools for discussing specific integration challenges and scenarios. Solutions are proposed, debated and prototyped, and ultimately, candidate specifications are published.

The OSLC organization today is simply a group of people from across the industry who share an interest in changing the status quo. They utilize modern on-line collaboration tools for discussing specific integration challenges and scenarios. Solutions are proposed, debated and prototyped, and ultimately, candidate specifications are published. OSLC has no membership fees or applications; no egregious bureaucratic processes; and no "purity tests" for participation. In fact, there is only one way to influence the work of the OSLC: to participate in it.

As we established OSLC, we knew that it would be critical that the intellectual property required to implement compliant products be open. However, it turns out that there is more than one way to be "open," and in order to choose the right way, we had to take a step back and review the business goals we were hoping to achieve.

The business value of open markets

The compact disc (CD) market flourished because Sony and Philips, the vendors who created the technology, licensed the necessary patents widely, reasoning correctly that both were better off sharing a huge market than owning a small one.

Since the beginning of the industrial age, smart companies have recognized the value of common specifications. Whether it's house bricks, screw threads, car parts, or electronic circuitry, markets work best when suppliers, purchasers, and end users can rely on the interoperability and interchangeability of parts regardless of source. Shared specifications promote competition, diversity of offerings, reuse of common components, and also innovation, because incremental improvements can be built on top of the existing foundations. Everybody is better off as a result: Sellers create more value, buyers receive more value, and innovators can add more value. The compact disc (CD) market flourished because Sony and Philips, the vendors who created the technology, licensed the necessary patents widely, reasoning correctly that both were better off sharing a huge market than owning a small one. By contrast, the adoption of both HD-DVD and Blu-Ray Disc, competing formats to succeed the DVD, was stalled for several years by the inability of either specification to become universally endorsed.[3]

Highlights

For many years, the word "standard" in IT was doublespeak for "dominant proprietary product."

And yet, for much of its early history, the IT industry shunned cooperation, with vendors preferring to protect proprietary fiefdoms rather than promote collaboration. For many years, the word "standard" in IT was doublespeak for "dominant proprietary product." [4] This practice continued despite the growing evidence that common specifications were better for everybody. Buyers benefit because vendors are forced to compete on quality, price, support, and pace of innovation. Vendors benefit because the market as a whole grows more rapidly -- so even though they have to compete with each other, the amount of business value available for them to target more than compensates for the loss of a small, protected niche.[5]

Less obvious but equally important: open markets create the opportunity for an ecosystem to emerge in which innovators can create compatible added-value add-ons, each finding their own niche in the ecology. For example, the definition of standardized slots and connectors in car dashboards provides customers with an enormous choice of entertainment options, both directly from the car manufacturer or as an "after market" upgrade. And of course, a car owner can upgrade the radio to replace the tape player with a CD player or the CD player with an iPod dock at any time, without having to replace the whole car! In fact, the creation of a value-enhancing ecosystem that allows many participants to benefit is often the single most important predictor for widespread adoption of a standard for technology.

The database market boomed with the adoption of the SQL open standard, in part because a variety of vendors could offer value-added tools for monitoring, tuning, and so on, and because skills such as database design became more portable.

There are numerous proof points in IT for both the benefits of open specifications and the importance of an ecosystem. The database market boomed with the adoption of the SQL open standard, in part because a variety of vendors could offer value-added tools for monitoring, tuning, and so on, and because skills such as database design became more portable. The personal computer market exploded when IBM established the PC as a dominant format, which allowed:

- Software authors to target one common base of customers rather than a dozen fragmented markets and
- Peripheral manufacturers to consolidate their costs around a single technical standard

In the world of development tools, agreement to use the Unified Modeling Language meant that modeling tool vendors no longer "competed" over fine distinctions between the shapes of lines and boxes, and instead competed on the effectiveness of their products in promoting business value derived from ways to leverage a common language. As a direct result, customers were able to choose

Highlights

from a wide variety of UML-compliant tools satisfying every need from the simplest to the most sophisticated, without having to risk locking themselves into the future prospects of a single technology vendor, and the modeling tools market flourished. More recently, the establishment of the Open Document Format (ODF) has allowed many different vendors to offer word processors and office suites in competition with Microsoft, whose proprietary format used to determine the "standard" in that market. Many user organizations are mandating ODF because an open standard guarantees them the crucial ability to access their own documents in the future without being governed by the decisions of one company.

"Open" comes in various flavors, ranging from a dominant proprietary format that is widely licensed to formal standard approved by an independent international body, and it's important to choose the right one for the business objectives at hand.

Today, the IT industry has, for the most part, embraced open markets, and it is rare to see a new technology initiative launch without a corresponding roadmap for establishing an open specification. However, as the examples above show, "open" comes in various flavors, ranging from a dominant proprietary format that is widely licensed to formal standard approved by an independent international body, and it's important to choose the right one for the business objectives at hand. The question that must be asked – and that is all too often is overlooked – is simply: What must be standardized in order to increase business value and decrease business cost?

Open Standards

The most common meaning of "open" concerns itself with standardization of the *behavior* of software. For example, the purchaser of a standards-compliant SQL database or a JEE application server can be confident that the software will produce the same essential results, regardless of the supplier. Of course, each vendor is still free to compete on quality, reliability, cost, support, and extensions of functionality beyond the standard; and to patent or keep as trade secrets uniquely effective methods of implementing that standard behavior. Thus, the benefits of open standards can be compared to the many standards for implementation that are commonplace in industry and engineering, such as DIN, ISO, ANSI, etc. Open standards gained broad adoption in IT during the 1980s, a period that saw the emergence of cross-industry organizations such as the OMG, Open/X and the IETF.[6]

Open Source

The 1990s saw the commercial rise of a new meaning of "open": open source.[7] In this model, the source code for an application is made freely

Highlights

There are few parallels outside of software to the value of open source, because of the unique nature of software.

available to anybody who wishes to use it, subject to certain restrictions (depending on the specific license used to publish the source code). The most common restriction is the requirement that any modifications to the source code must be shared back with other users under the same terms. Unlike open standards, which promote common behavior, open source promotes common implementations. There are few parallels outside of software to the value of open source, because of the unique nature of software. One loose analogy is to classic books that are out of copyright: the text of that book might be available in numerous editions from different publishers, including hardback, paperback, annotated, collected with other works, and so on. But even this analogy fails capture the true value of open source software, much of which comes from the unique flexibility with which software can be modified, adapted, and extended. In fact, some of the most commercially significant open source projects have been those that promote a common "platform" that allows innovators to add value on top without having to constantly recreate the common parts, and furthermore allows each innovation to enhance the value of all the others in a virtuous spiral. Individual innovations may themselves be open source, or use conventional commercial licenses.[8]

Open interfaces

Open interfaces are analogous to the electricity supply: you don't care how the electricity was generated, whether by wind, coal, or nuclear power. You only care that when it reaches your power receptacle, it provides the expected voltage.

Most recently in the current decade, a new meaning for "open" has gained prominence: open interfaces. This model came about from the realization that often, companies want to integrate a wide variety of tools or processes that need to be able to exchange information, even though each may not know or care what the other endpoint does with that information. To understand how powerful this idea is, think of it as being analogous to the electricity supply: you don't care how the electricity was generated, whether by wind, coal, or nuclear power; you don't care what path it took to get to your office or what voltage transformations it went through on the way. You only care that when it reaches your power receptacle, it provides the expected voltage and frequency. Open interfaces in IT provide the means for applications to act like "utility suppliers" of information to each other.

One of the earliest and most widely known successes for this model was in the arena of news "syndication." A very simple standard[9] was established that allowed servers to publish a series of changes as a "feed," often as news headlines and associated stories, and allowed clients to aggregate many different

Highlights

feeds and present them to a user. The servers are free to choose any implementation, frequency of publishing, and format of their contents that they want, as long as the description (in other words, the interface) follows the standard. Conversely, users can choose from a huge variety of ways to view, organize, and filter their chosen feeds. Today, a vast array of information sources far beyond traditional news stories — ranging from personal blogs to system maintenance logs — can be consumed as feeds.

The openness of OSLC

Faced with all these choices, we had to ask ourselves: what kind of "open" should OSLC be? As we considered the business drivers, we concluded:

- Demanding that user companies reskill and retool, or that participating vendors substantially redevelop their existing products, was not an option. Therefore, the common implementation approach implied by open source was not appropriate.
- Software development and delivery involves not only many different tools but also many different approaches, so standardizing the behavior of even the most commonly used types of tools would make little difference to the overall challenge. Therefore, the common behavior approach implied by traditional open standards was not appropriate.
- In order to support the diverse array of tools, processes, and business needs – including many we don't even yet know about – we realized that the most important thing to agree on was the interfaces between different tools.

We realized that the most important thing to agree on was the interfaces between different tools.

By focusing on agreement about the interfaces, we could allow each tool provider to participate without having to predetermine all the ways that users might integrate their particular tool, or requiring close relationships with all the other tool vendors a user might choose. And we could allow each tool user to integrate their choice of tools in the way that made the most sense and delivered the most business value in their own circumstances.

By focusing on agreement about the interfaces, each tool provider participates without having to predetermine all the ways that users might integrate their particular tool, or requiring close relationships with all the other tool vendors a user might choose.

We also determined that OSLC interfaces had to satisfy a number of rigorous requirements, including the following:

- *Neutral*. The interfaces must be both technology- and architecture-neutral, and could not favor the implementation details of any particular vendor's existing product.

Highlights

- *Adaptable*. It must be feasible to retrofit OSLC interfaces to existing products. without extensive reengineering of products currently in use.
- *Universal*. This was the most critical and most challenging requirement. Put simply, it means that any asset, resource or artifact used in the software development and delivery lifecycle must be able to reference and associate itself with any other, regardless of type, location, or implementation.

In OSLC, we have adopted exactly the same technical and architectural approach as the Web itself.

As noted earlier, as we considered these requirements, we realized that there is already an outstanding example that satisfies all of these needs: the Web. All links are URLs (universal addresses), and they have the critical benefit that any Web page or online resource can point to any other, without depending on any knowledge of what is at the other end of the link. In OSLC, we have adopted exactly the same technical and architectural approach as the Web itself. OSLC specifications depend on a small set of universal principles:

- Any asset can refer to any other asset using just one mechanism, namely a URL that identifies the location of that asset. Conversely, in order to make its assets available to any other OSLC-compliant tool, a tool need only expose an appropriate URL for each asset.
- The list of operations that one tool can perform against another is very short, and is also universal. For instance, a tool can request a resource from anywhere: directly from another tool, an asset management repository, a content management system, a configuration management system, and so on.[10] It can modify that resource. And it can then return the modified resource to the original tool, which assimilates the changes. If another tool that provides the same type of resources is substituted, the integration should not change.
- The format, or representation, of a resource is independent of the internal format, technology, or representation of any particular provider. Furthermore, the requesting tool can ignore any part of the representation that it doesn't understand or isn't concerned with (provided it preserves the information when it passes it on or passes it back).

When tools expose OSLC interfaces, the assets that they house can be linked and used just like a "Web" of software development and delivery resources.

In other words, when tools expose OSLC interfaces, the assets that they house can be linked and used just like a "Web" of software development and delivery resources. One powerful consequence of this is that OSLC integrations are "loosely coupled" and independent of the target tool. For example, if one part

Highlights

of your testing organization is using Tool A from one vendor while another part is using Tool B from another vendor, your requirements analysts can link their requirements to test cases in exactly the same way, regardless of which test team they are collaborating with. And if the requirements analysts decide to switch to a different requirements tool, it does not disrupt their collaboration with the testers, provided that the new tool also implements the OSLC interfaces.

The OSLC community

We recognize that customers frequently mix tools from several vendors, as well as open source projects and home-brewed utilities.

The community approach that we are taking at OSLC is just as critical to its success as the technology neutrality described above. We recognize that customers frequently mix tools from several vendors, as well as open source projects and home-brewed utilities. Experience proves that any vendor-specified set of APIs will invariably privilege one vendor's tools and disadvantage any others' – or at the very least, make other vendors suspect this will be the case. Vendor-specified APIs also leave many partners companies struggling to catch up as the vendor evolves the API independently or in collaboration only with its closest partners, leading to suspicion and reluctance to participate by competing vendors. The result has been the fractured and unsatisfactory integration landscape of the past, as described at the beginning of this paper.

To promote community involvement and ensure transparency, all of the work of OSLC is taking place in the open, at http://open-services.net.

Therefore, to promote community involvement and ensure transparency, all of the work of OSLC is taking place in the open, at http://open-services.net. There, OSLC is organized by domain-specific working groups in areas such as Change Management, Requirements Management, Quality Management, Project Estimation, and Metrics. Each work group defines and prioritizes the scenarios, describes the scope of each iteration, and writes the specifications for interacting with tools in their domain of interest.

So that the overall objectives of OSLC are met within each working group – and to avoid the creation of silos that do not interoperate – the project leads from the OSLC working groups coordinate their work across the domain areas. The project leaders also collaborate on topics of common interest and in sharing design best practices that they discover in the course of their work.

To ensure that the content developed for OSLC is broadly consumable, OSLC has adopted two important intellectual property policies. Firstly, all content posted to the site is covered under a Creative Commons license (see Box 2), which has very

Highlights

liberal usage rights. This license allows anybody to freely use the specifications without any license fee or restrictive agreement. Companies don't even have to be members of OSLC to use the specifications: they are freely available to everybody.

Secondly, all contributors to an OSLC specification publish a patent non-assert covenant, promising not to assert any "necessary claims" against implementations of the specifications. These policies have been important in securing the participation of both open-source and commercial concerns. OSLC participants include individuals from software vendors, open source projects. systems integrators, industry IT teams, and the academic community.[11]

Conclusion

Overcoming the challenges of tool interoperability isn't easy, but it is important to the industry as a means of breaking software delivery resources out of the IT silo and making them more accessible and interoperable with the broader business processes.

Overcoming the challenges of tool interoperability isn't easy, but it is important to teams that apply tools as they endeavor to differentiate their businesses. It's important to tool providers who face demands for an ever-increasing set of pairwise integrations. And we would argue that it's important to the industry as a means of breaking software delivery resources out of the IT silo and making them more accessible and interoperable with the broader business processes. OSLC seeks to change the game, but in a practical way, learning from the lessons of the past and building on the success of the Internet.

One thing that is clear: OSLC gets better with participation. The more people involved, the better chance we have to establish real collaboration across the lifecycle. If you have something to contribute, we encourage you to join in at http://open-services.net.

Highlights

The most basic technical capability is the need to create and manage navigable relationships between "resources" or assets. On the Web, this problem has been solved.

Appendix A: The technical architecture of OSLC

To collaborate across the software lifecycle, the most basic technical capability is the need to create and manage navigable relationships between "resources" or assets. Resources include everything from requirements to code to test cases to configurations. Complicating matters, some of these resources can be complex formatted documents such as a requirements specification; others are collections of other resources, such as a configuration; others are artifacts created by the process itself, such as a build log. All of these different kinds of resources need to be treated in a uniform way if tools are to be able to link their assets to each other without hard-coded foreknowledge of the target endpoint.

Identifiable and addressable: The value of URL-accessible data

On the Web, this problem has been solved. All links are URLs (universal addresses), and they have the critical benefit that any Web page or resource can point to any other, without depending on any knowledge of what is at the other end of the link. In OSLC, we wanted to create a similar means of relating resources that, like the Web, satisfied three basic criteria:

- It must be vendor- and technology-independent: it must provide a way for any vendor's representation of a test case, for instance, to point to any other vendor's requirement.
- It must be location-independent: resources are scattered across multiple repositories and tools, possibly widely or even globally dispersed. And those resources can move from time to time.
- It must be able to solve the problems we don't yet know about, not just the ones we do: the future will certainly bring new types of asset into the lifecycle, as well as new styles of integration and aggregation. Like the Web itself, the linking mechanism must be flexible enough to adapt to change.

Identification and location are the most fundamental parts of any digitally based interoperability story. Like the Web, every resource in OSLC has a unique address that tells any other tool how to locate it – in other words, a URL. It is impossible to overstate the fundamental importance of having URL-accessible data. This is what makes an application's data *identifiable and addressable* from any other application, in a completely technologically neutral fashion. A tool that needs to refer to a resource used by another tool, such as a test tool with a test case that references a requirement, need store

Highlights

In order to create more useful integrations, we need one more thing: a standardized "interface," or common set of services, that are available for resources. Again, the Web has provided an answer for us.

only the URL that identifies that resource, just like one Web page linking to another. By contrast, many existing ALM tools rely on proprietary naming schemes that require deep knowledge of the tool's technology to resolve.[12]

Having a uniform way to identify and locate resources is essential, but by itself merely allows us to establish pointers or references from one resource to another. In order to create more useful integrations, we need one more thing: a standardized "interface," or common set of services, that are available for resources. Again, the Web has provided an answer for us that we leverage in OSLC: a small set of actions that can be used identically from any tool to any other tool, regardless of the type of resource or the technical implementation. OSLC uses a style of integration called REST to fetch or modify any resource, regardless of its type or location that ensures that tools remain independent of each other's implementation details. In a RESTful architecture, clients initiate requests to servers using a small set of general-purpose services. Servers process those requests and return appropriate responses.[13]

Representations

In OSLC as on the Web, the notion of a resource is very general. It could be a requirement, a work item, or even a Web page that enables a user to create or select from a list of resources that meet certain criteria. These are all resources, identified by their URLs.

URLs, along with a RESTful architecture, allow us to locate and access such a resource, which is often useful by itself, but, by design, these URLs do not provide a tool with any information about what's "inside" the resource: the content is still unknown. For example, a requirement could be represented by a text document describing the requirement, an image showing a screen mockup, an XML document defining the attributes of the requirement, or any one of many other representations. A test case management tool that wants to verify the existence of a test-case for every requirement need not understand the requirement's contents; it need only confirm its existence and location (i.e., its URL). When the tool's users want to see or edit the contents of the requirement, it functions exactly like a browser: it navigates the link to retrieve the resource and hands it to an appropriate tool that understands the resource type.[14]

Highlights

Much of the work in OSLC working groups is to define agreed-upon resource representations. With this information, any tool can examine the common elements of these resources; this allows much deeper integration between tools.

Although we value this flexibility, we can do more with the resources when we know some details about their format. Therefore, much of the work in OSLC working groups is to define agreed-upon resource representations. With this information, any tool can examine the common elements of these resources; this allows much deeper integration between tools. For example, a tool could view and modify the description of any lifecycle resource. A quality management tool might refer to a requirement stored in a requirements management tool and be able to reflect whether associated test cases pass or fail. A configuration management tool might flag tests stored in a test case management tool as needing to be run during the next regression test because relevant code modules have been changed. A management console might pull information from a wide variety of tools to produce a consolidated view of project progress and status. And, just like the Web, the retrieved representation of a resource might contain links to further resources: for example, a defect implicates a code module that is tested by a test case that satisfies a requirement. Each of these links exploits the same mechanisms: URLs and uniform services.

Appendix B: Creative Commons Licenses and Copyrights

OSLC chose to publish its specifications under Creative Commons licenses so that everybody could adopt those specifications with confidence. A Creative Commons license is a mechanism that allows a copyright holder, such as the authors of an OSLC specification, to renounce certain rights that they would normally have under copyright law, while retaining others. This mechanism is what allows any implementer to take and use the published OSLC specifications without risk that some kind of license claim will be enforced against them.

Critically, a Creative Commons license cannot be revoked: for example, if a vendor implements the published Change Management 1.0 specification, the copyright owners can never revoke that vendor's permission to use that specification in the ways initially permitted.

More information about Creative Commons is online at http://creativecommons.org

Endnotes

[1] In July 2008, Google announced that its Web search surpassed one trillion unique URLs.

[2] Some commercial content providers have objected to a practice known as “deep linking” because it bypasses their ability to control a site visitor’s access path or to enforce subscription payments. However, this is an issue of business models, not of technology.

[3] Ironically, the DVD standard was itself almost derailed before it even started by an eerily similar format war. The intervention of IBM, which rallied other computer vendors to agree to boycott both camps unless they compromised on a unified specification, is a little-known story in the success of DVD.

[4] Such situations are so common that the IT industry needs a term for these “de facto” standards, as contrasted with “de jure” standards, which are standards established by an independent body and available equitably to any market participant.

[5] Imagine how much the value of mobile phones would be diminished if, for example, customers of Sprint could only call other customers of Sprint, and customers of Verizon only other Verizon customers. Extraordinary as it now sounds, that is in fact exactly how email worked in the early days of online access. Subscribers to one online service could only email customers on the same service. Even today, the instant messaging services provided by various Web portal providers pursue this same “walled garden” approach.

[6] Some basic standards have existed from the earliest days of IT, especially in areas strongly influenced by IT’s highly standardized cousin, telecommunications. For instance, ASCII dates back to the 1960s, and other early examples include the programming languages COBOL and FORTRAN. However, it was not until the 1980s that higher levels of system behavior were widely standardized.

[7] The open source movement had been influential in academic circles for at least two decades before it began to impact commercial software adoption, for example at MIT in the GNU project. However, the impact on commercial IT rose dramatically with the creation of Linux in the early 1990s and the Apache Web server in the mid 1990s.

[8] One highly successful example of this model is the Eclipse Foundation: see *http://eclipse.org/*

[9] In fact there are two competing standards that are widely used today, RSS and Atom. Although they differ in detail, both serve the purpose described here, and both share the distinction of standardizing only the interface between servers and clients. The name RSS is often used casually to include both standards.

[10] Technically speaking, this is a “representation” of the resource. See Appendix A for a more detailed explanation of this distinction.

[11] Learn more about OSLC community members at *http://open-services.net/html/Snapshot.html*

[12] For example, one tool may require you to know the case-sensitive name of a requirement. Another may require the use of an internally-generated serial number. A third may provide a persistent “handle” that can only be decoded by calling a proprietary API. In order to make use of any of these reference mechanisms, the referring tool must know both what kind of reference it is dealing with, and also what tool it refers to.

[13] For a more rigorous and complete definition of REST, see Roy Fielding’s PhD dissertation, in particular Chapter 5, available online at ***http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm***.

[14] For example, if the user of a browser clicks on a link to a PDF document, the browser will ask Adobe Acrobat to handle the document that is returned from the server. If the link points to a document in OpenDocument Text (ODT) format, the browser will use whatever tool is available that supports ODT.

© Copyright IBM Corporation 2009

IBM Corporation

Software Group

Route 100

Somers, NY 10589

U.S.A.

Produced in the United States of America

December 2009

All Rights Reserved

IBM, the IBM logo, ibm.com and Rational are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Other company, product, or service names may be trademarks of IBM or other companies.

A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml

The information contained in this document is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided “as is” without warranty of any kind, express or implied.

RAW14207-USEN-00