Agility @ Scale - Enabling Collaboration across Silos

Erich Gamma
IBM distinguished engineer
IBM rational zurich research lab

First Assignment: Eclipse

- A tools integration platform
  - Scalable
  - Easy to extend
  - Enable a tools ecosystem
- Goal: Built to last

Inspiration: how buildings last

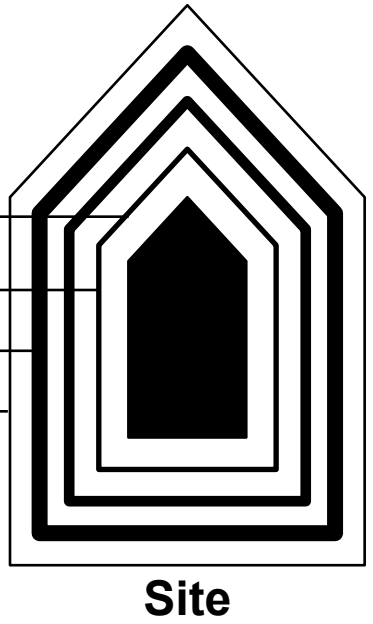
- **Stewart Brand: how buildings learn – what happens after they're built**

  stuff: furniture

  services: electrical, plumbing (7-15y)

  structure: foundation, load bearing walls (30-300y)

  site: geographical setting (forever)



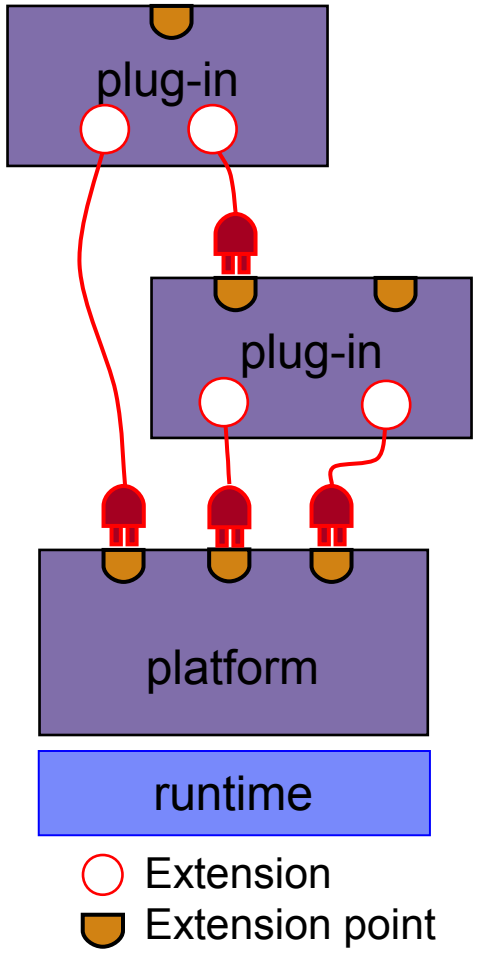
- **layers:**
  - evolve at different rates during the life of a building
  - shear against each other as they change at different rates
  - an adaptive building must allow slippage
  - ➢ a building that lasts is adaptive and can change over time
  - ➢ lasts for generations without total rebuilding

structure foundation

- the eclipse plug-in architecture
- everything is a plug-in
  - simple and consistent

eclipse plug-in architecture

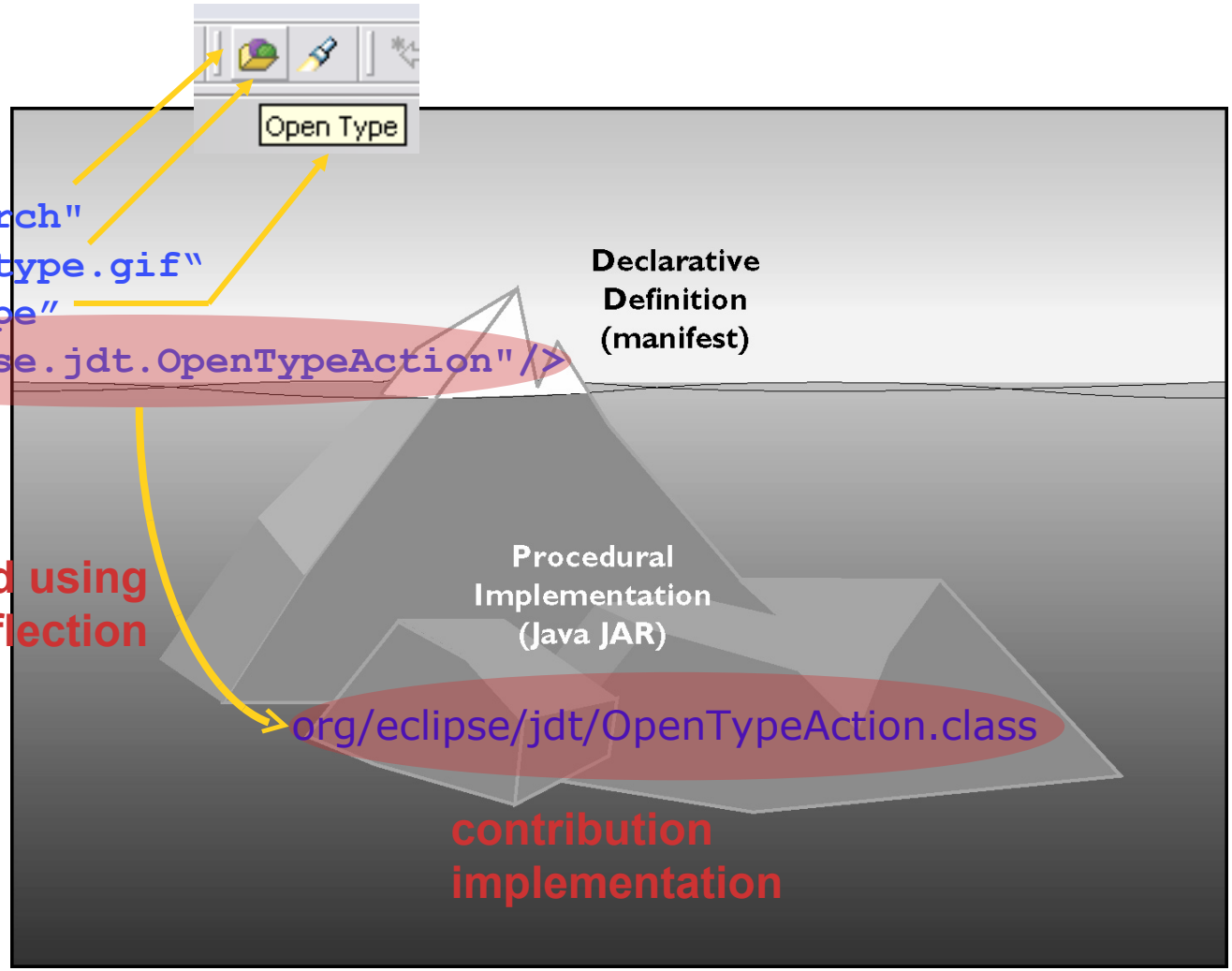
- **plug-in == component**
  - set of contributions
  - smallest unit of Eclipse function
  - details spelled out in plug-in manifest
- **extension point** – named entity for collecting contributions
- **extension** – a contribution
  - Example: a specific spam filter tool
- **runtime** – controls and manages contributions



scalability

user visible appearance

```
<action
    toolbarPath="search"
    icon="icons/opentype.gif"
    toolTip="Open Type"
    class="org.eclipse.jdt.OpenTypeAction"/>
```



lazily instantiated using reflection

services plumbing: APIs

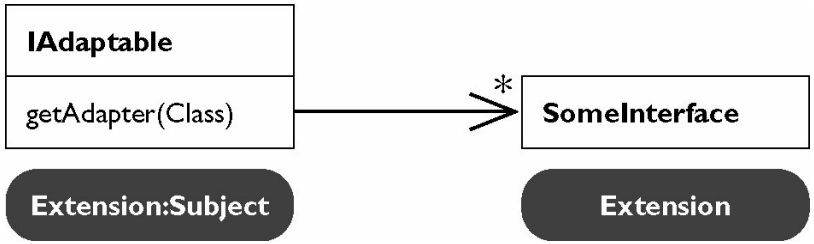
- Plug-in dependencies through APIs
- define APIs for stability
  - binary compatibility is highest priority

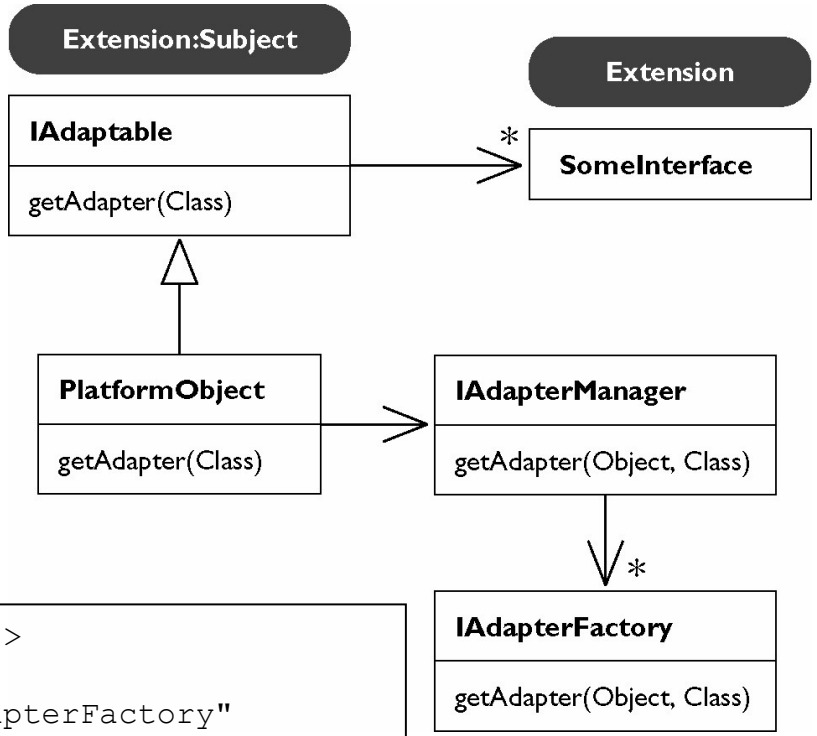
APIs first

- APIs don’t just happen; we need to design them
- specifications with precisely defined behavior
  - what you can assume (and what you cannot)
  - it works ≠ API compliant
  - documented classes ≠ API
- must have at least one client involved, preferably more

extension interfaces: IAdaptable

- adding interfaces to existing types
- Interface negotiation






```
<extension point="org.eclipse.core.runtime.adapters">
<factory
   class="org.eclipse.jdt.internal.ui.JavaElementAdapterFactory"
   adaptableType="org.eclipse.jdt.core.IJavaElement">
   <adapter type="org.eclipse.ui.IPersistableElement"/>
  …
</factory>
```

I*2 extension interfaces

- add new methods in extending API interface with extension interfaces
  - avoids breaking existing implementors of an interface

```
public interface IActionDelegate { … } // original interface

public interface IActionDelegate2 extends IActionDelegate {
   void dispose();
}
```

```
 if (d instanceof IActionDelegate2) {
   IActionDelegate2 d2 = (IActionDelegate2) d;
   d2.dispose(); // call new method
 }
```

Key Lessons

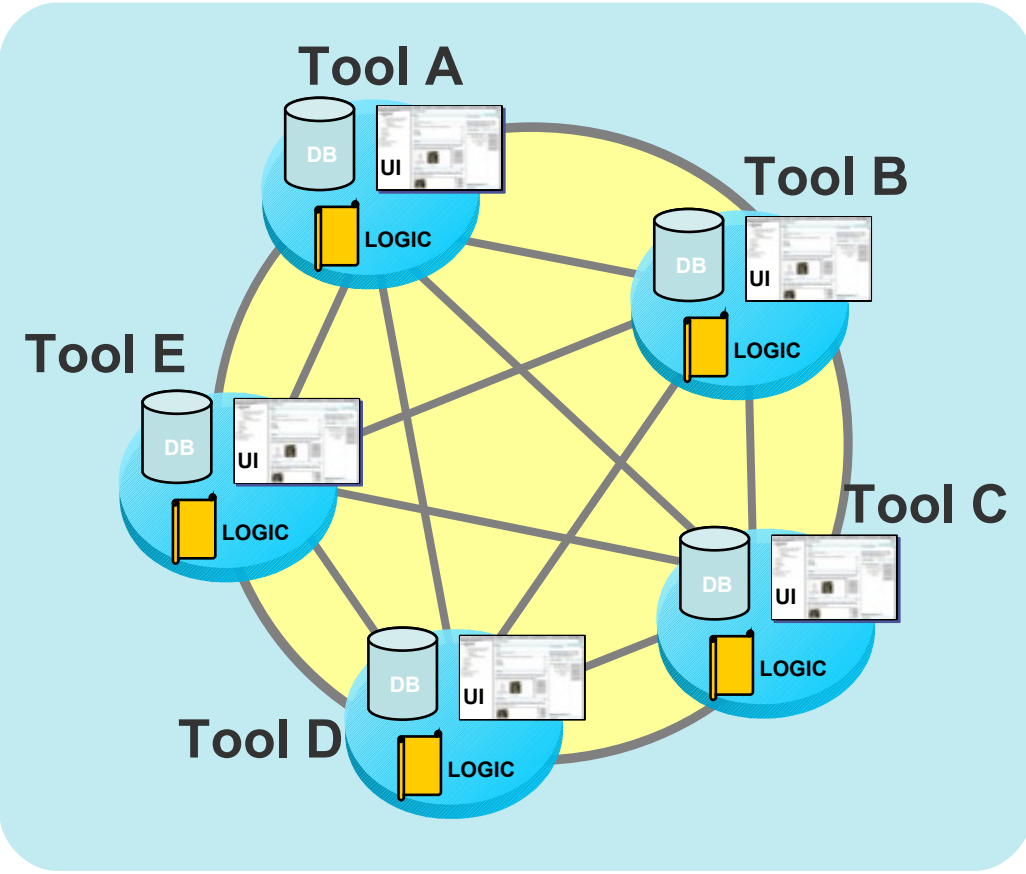
- **Modularity matters**
  - **Everything is a plug-in**
  - **“no exceptions”**

APIs are a huge commitment

- we would rather provide less API than desired (and augment) than provide the wrong (or unnecessary) API and need to support it indefinitely
- the tyranny of stable APIs
  - API layers…
- the challenge of product developers
  - which API level does our product require and support
    - n–1, n-2

Next assignment: A Tool Integration Platform

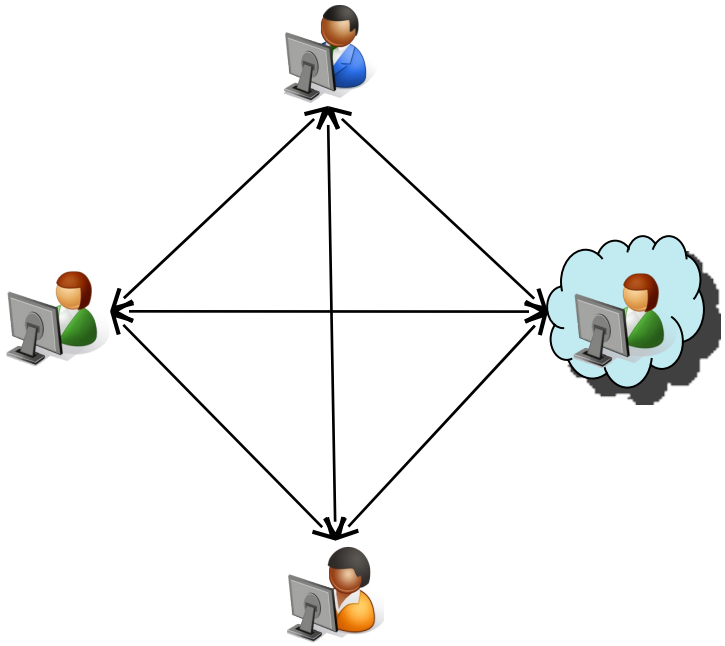
- Integrate many tools
  - **Heterogeneous environments** that are flexible for partners and suppliers
  - Acquisitions raise expectations for product **integrations**
- Global Connectedness
  - **Distributed** development, cross site product development
- Lifecycle / Agile Methods
  - **Flexible** tools and process



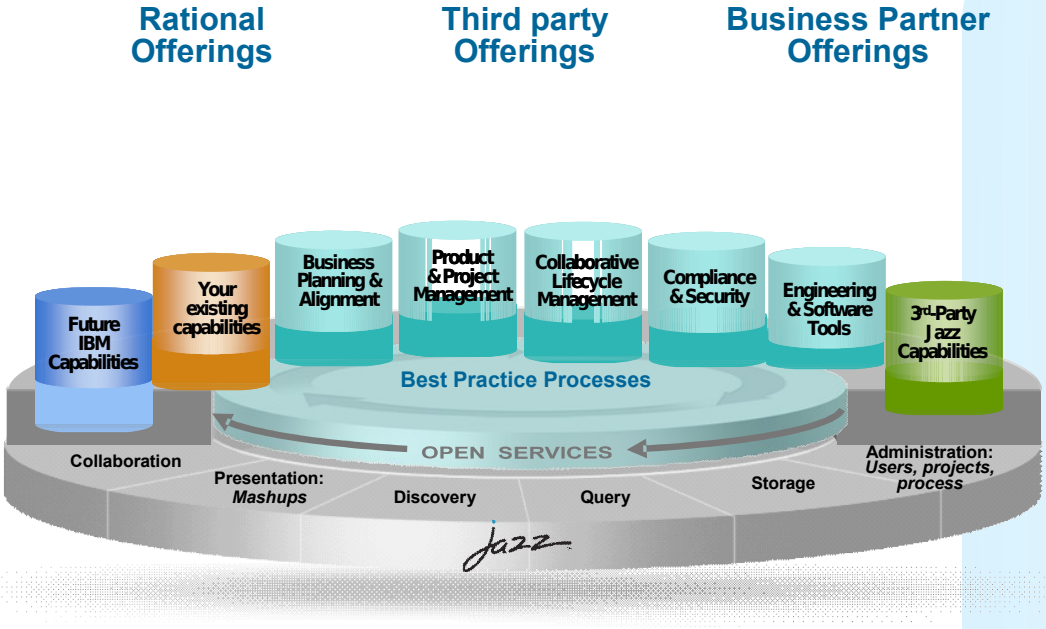
Traditional Tool Integration. Ouch.

- N2 possible point-to-point connections
  - Limited coverage
- Closed APIs
  - Vendor lock-in
- Tight Coupling
  - Dependence on internal structures
- Lockstep upgrades
  - Version incompatibilities

- Need something better…



Jazz is a platform for transforming software delivery



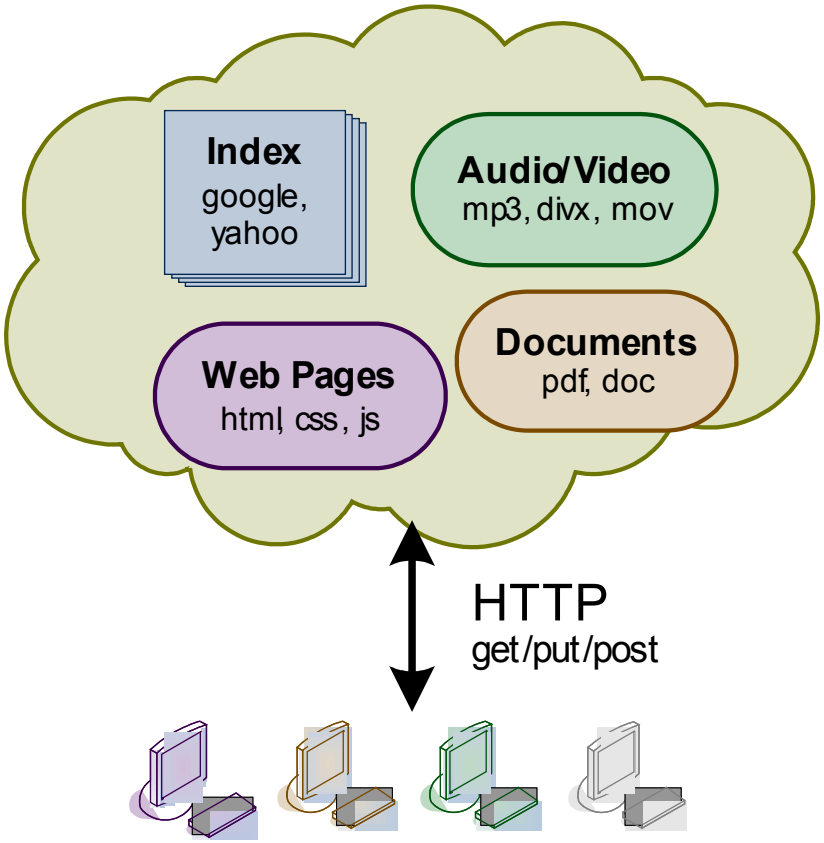
Jazz is…

- Our vision of the future of systems and software delivery
- A scalable, extensible team collaboration platform
- An integration architecture enabling mashups and non-Jazz products to participate
- A community at Jazz.net where Jazz products are built

Jazz is a platform for *transforming how people work together* to deliver greater value and performance from their software investments.

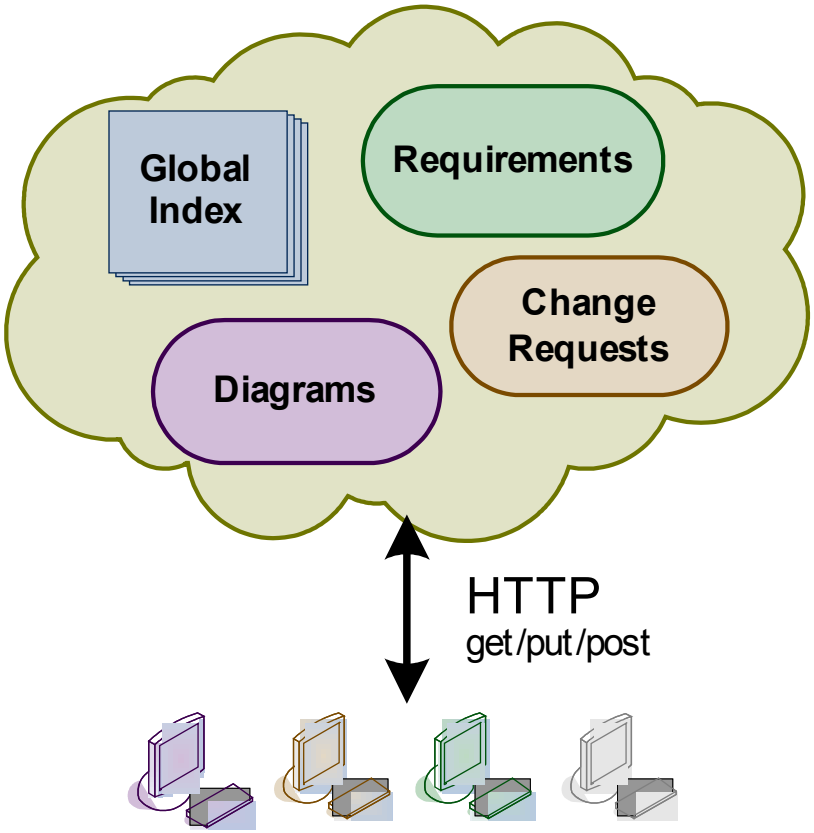
Inspiration: the Internet

- Amazingly scalable
- Integrates information on a massive scale
- Infinitely extensible
- Collaboration on unprecedented scale
- World-wide information visibility



How does this work?

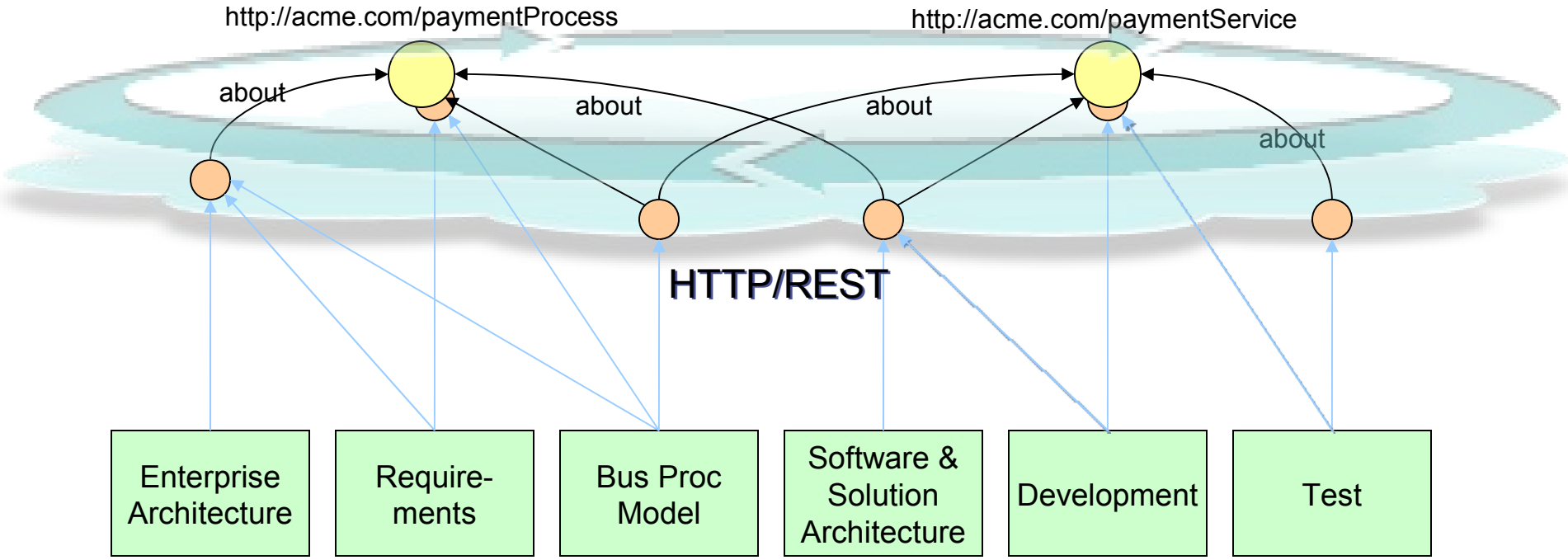
- All data are resources with URLs
- Resources have representations
- Representations are specified independently of tools
- Links are embedded URLs
- Tools (multiple) access data through HTTP get/put/post/delete



Jazz architectural principles

- Jazz separates the implementation of tools from the definition of and access to the data
  - Data semantics do not rely on "secret knowledge" embedded in product code.
- Jazz can access and integrate data where it resides
  - Jazz does not need to import and export data between tools or repositories
- Jazz assumes an open, flexible, distributed data model.
  - Jazz does not assume that there is a single data model that is centrally managed, nor that each tool needs to understand the entire data model in order to participate.
- Jazz allows tools to be implemented in any Internet-aware programming language or platform.
  - Jazz does not impose an implementation framework tied to a particular language or technology platform
  - Provide optional toolkits to aid in tool implementation

Data Integration – the new way – “www linked data”



Architectural Rules

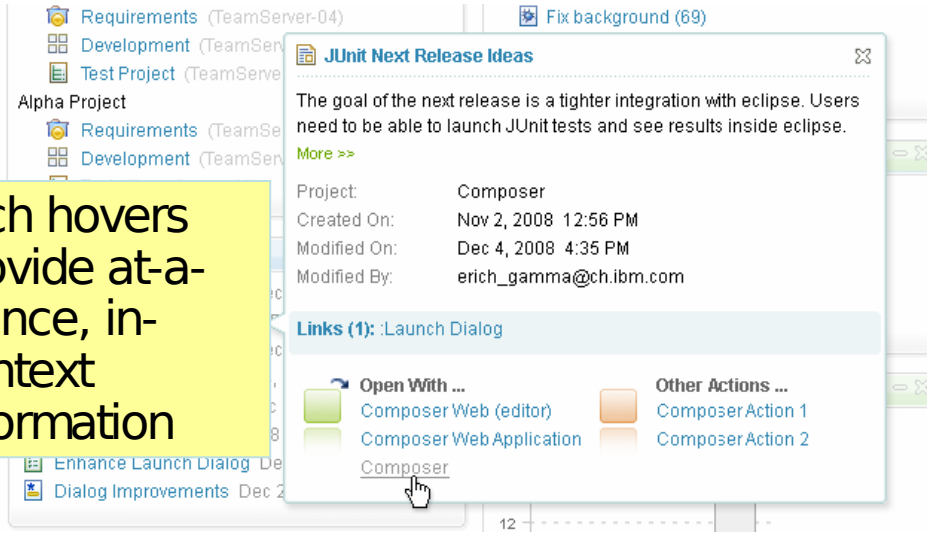
- R1: Independent upgrade
- R2: Rich Integration
- R3: Limited application coupling
- R4: Open world

R1: Independent upgrade

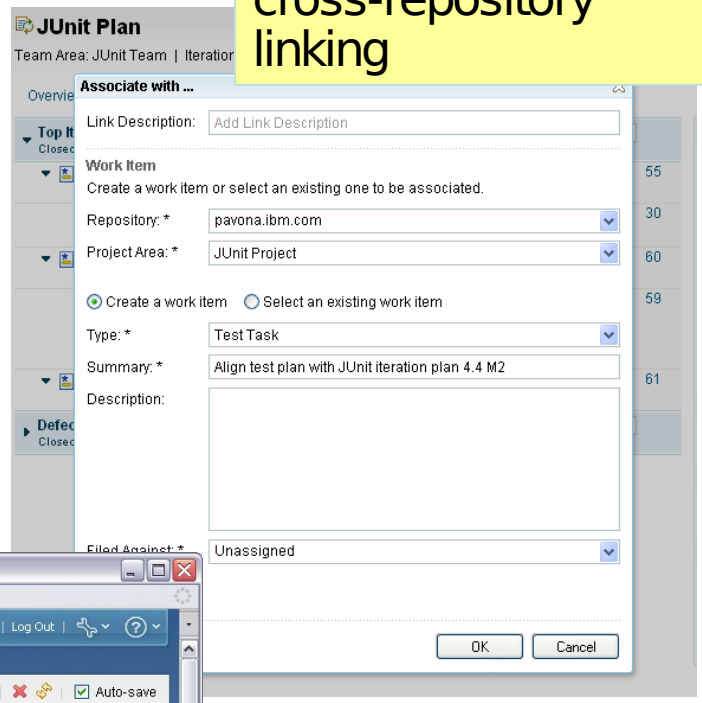
- Customers must be able to upgrade their products **one at a time** in the order of their choice
  - product teams must commit to managing their dependencies so that this will always be the case
- Easy to say; easy to understand; highly motivational
- Smooth upgrading is a corollary
  - customers must not feel that they are losing/breaking their applications (or application data) as a side effect of upgrading any of their products.
  - Client - server compatibility issues are included here.

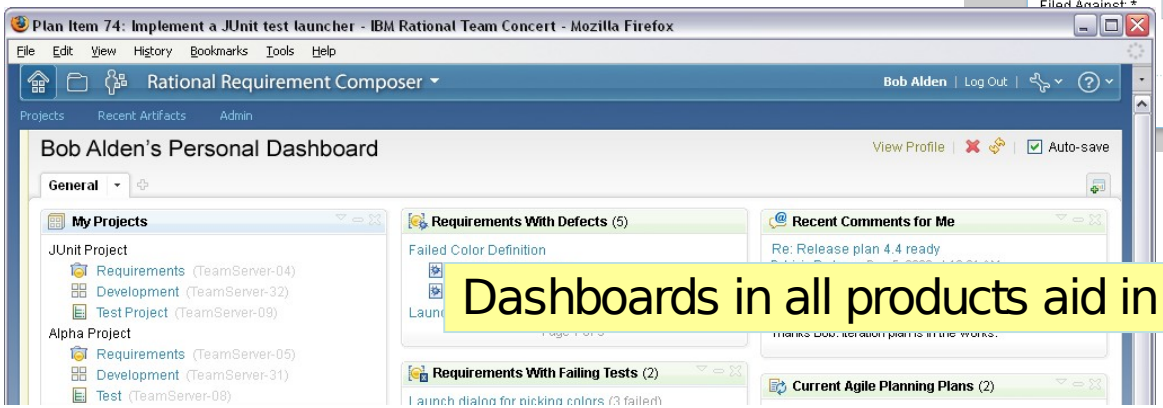
R2: Rich integration (with loose coupling!)

Rich hovers provide at-a-glance, in-context information

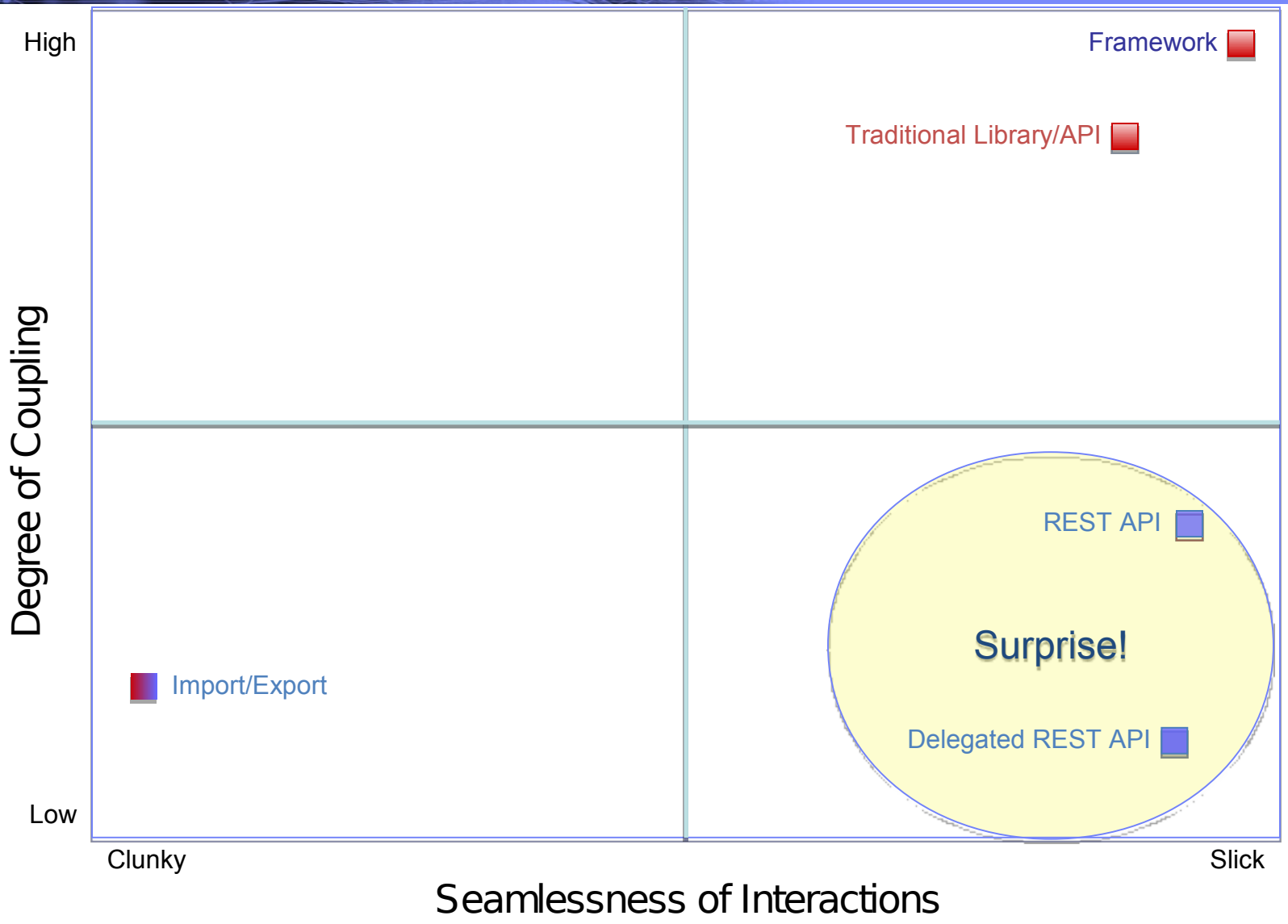


Link Dialogs enable cross-repository linking





Dashboards in all products aid in transparency


High
Framework
Traditional Library/API
REST API
Surprise!
Import/Export
Delegated REST API
Low
Clunky
Slick
Degree of Coupling
Seamlessness of Interactions

R3: Limited application coupling

- Applications will depend on few other applications.
- If we're not careful, we get caught in the dependency web
- Yet, applications need to interact

R4: open world

- New products can be integrated after the fact, and their capabilities are reflected in the user and programmatic interfaces
- Don’t assume you know everything up front

Open Services for Lifecycle Collaboration

An initiative aimed at simplifying tool integration across the software delivery lifecycle

Open Services for Lifecycle Collaboration

Barriers to sharing resources and assets across the software lifecycle

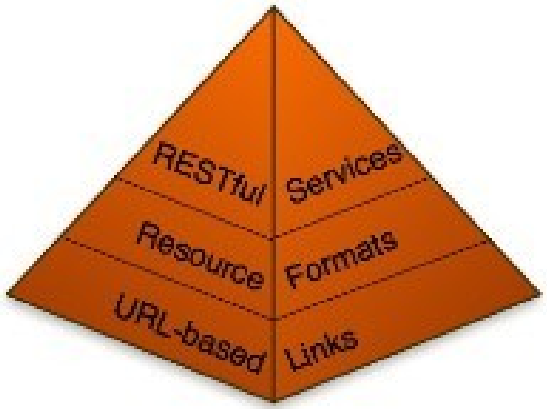
- Multiple vendors, open source projects, and in-house tools
- Private vocabularies, formats and stores
- Inextricable entanglement of tools with their data

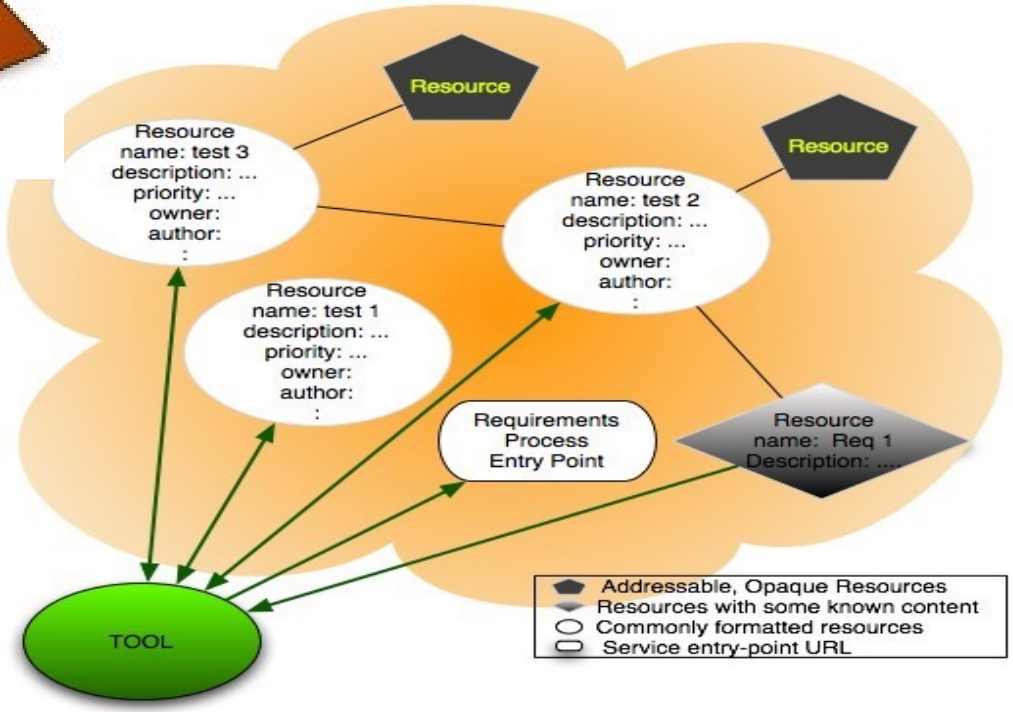
- Specifications for sharing lifecycle resources
- Inspired by Internet architecture
  - Loosely coupled integration with “just enough” standardization
  - Common resource formats and services
- A different approach to industry-wide proliferation

Open Services for Lifecycle Collaboration

Putting the approach into practice

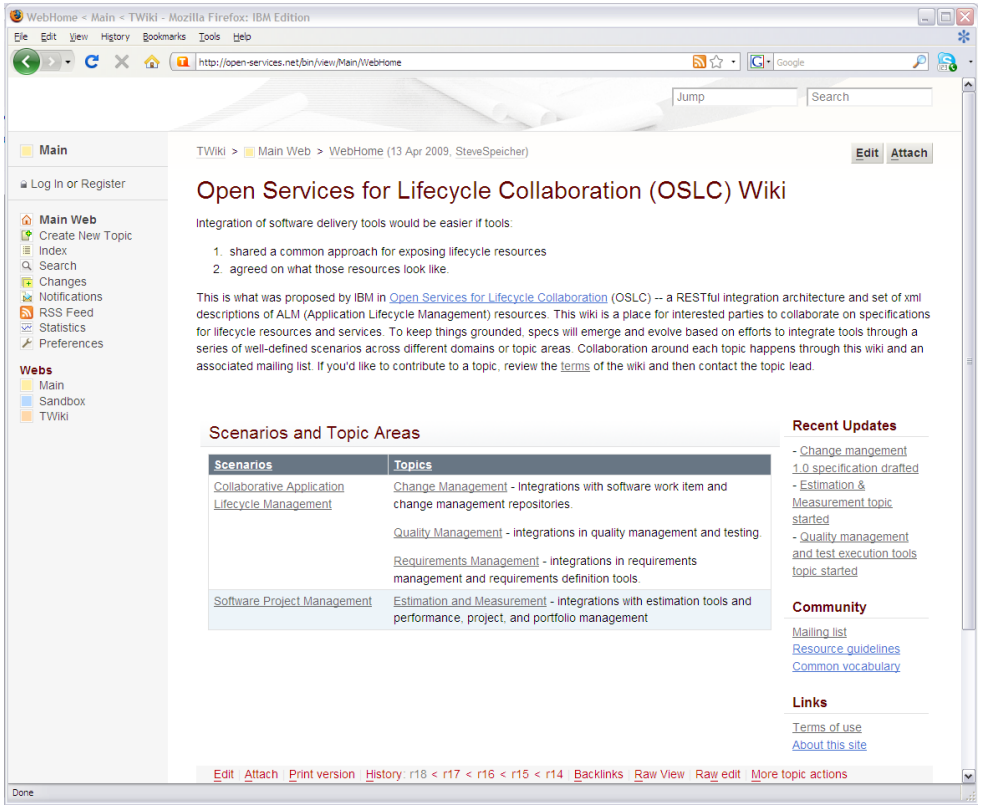
- **Step 1: Internet URLs for resources**
- **Step 2: Shared resource formats**
- **Step 3: Shared resource services**





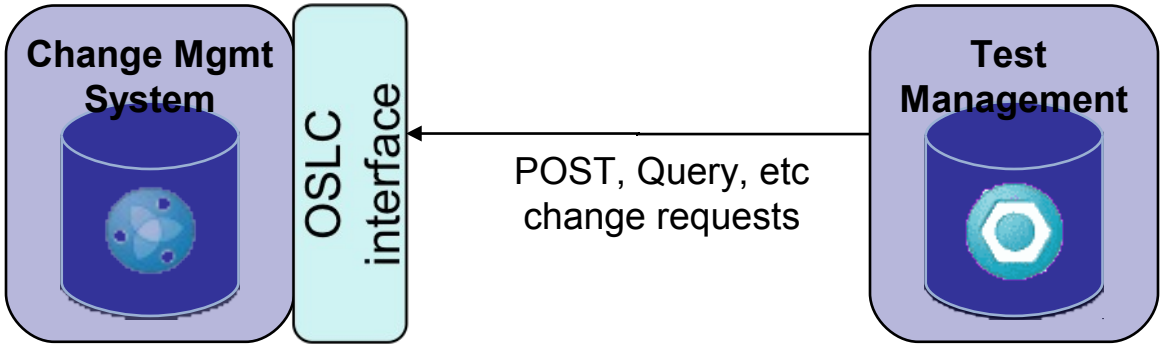
Community: open-services.net

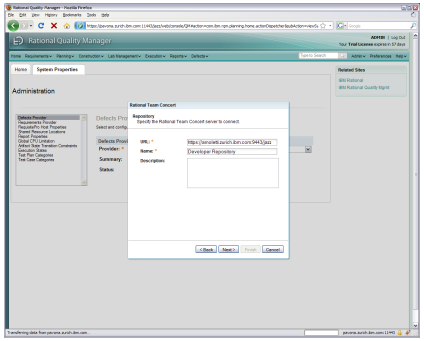
- Started in 2008
- Open community contribution
- Scenario driven...a minimalist approach
- Divided into focus areas
  - Change Management
  - Quality Management
  - Estimation & Measurement,
  - Requirements Management, …
- Solving integration in the open



OSLC at Work

Loosely coupled integration with "just enough" standardization





- Spec dictates the bare minimal aspects of defect
- QM system posts "seed data"
- QM system gets URL of form; delegates back to CM system

QM system can interface with *any* OSLC-compliant change management system

Styles of Integration

- HTTP REST API – “Rich” style
  - Web technologies – pervasive support across languages and Operating Systems
  - Resource-oriented – requires agreement on the resource representations
  - Careful resource design can avoid “closed world” assumptions
  - Exposes details of the data in resource representations
  - Can leverage client libraries, but does they are outside of the API boundary
- HTTP REST API “Delegated”/Widget Style
  - Relies on discoverable URLs for services
  - Minimizes dependencies: delegates back to application
  - Introduces out-of-bands communication between delegated form and host application

OSLC Specification http://open-services.net/bin/view/Main/CmSpecificationV1

Document
CM RESTful Services
CM Change Request Resource Defin
CM Simple Query Syntax
CM JSON Format
CM Delegated Resource Selection ar
CM Service Description

Resource URIs and Methods

Resource	URI	GET	POST	PUT	DELETE	Description
Collection of Change Requests	{CR Collection URI}	Y	*	N	N	A collection of change requests
Change Request	{CR URI}	Y	N	Y	Y	An identifiable change request, by a permanant URI

* - the collection MAY support creation on its URI, see Create a new Change Request

N - in the HTTP verb column indicates that a Service Provider MUST return a 405 Not Supported response

For a complete list of HTTP Response Codes

URIs for working with Change Requests

The following table outlines the key items that are exposed in the Change Management Service Discovery Document. Details of each of these capabilities will follow in subsequent sections.

Purpose	Discovery Element	URL*	Section	Support
Resource Creation	<factory>	{Resource Creation URL}	Create a new Change Request	REQUIRED
Resource Query	<simpleQuery>	{Simple Query URL}	Get a Collection of Change Requests	REQUIRED
Resource Selection UI	<selectionDialog>	{Selection Dialog URL}	Resource Selection	REQUIRED
Resource Creation UI	<creationDialog>	{Creation Dialog URL}	Resource Creation	REQUIRED

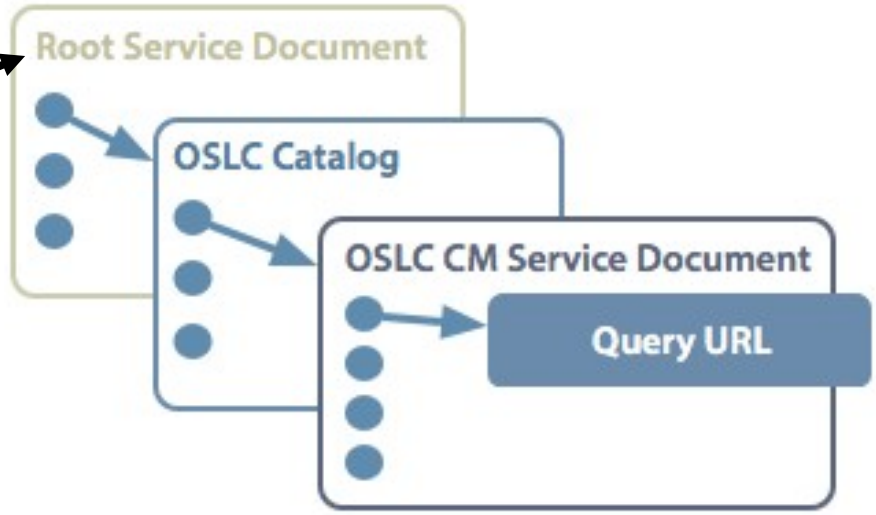
Retrieving a Defect

```
GET https://rtc.com:9443/jazz/resource/itemOid/com.ibm.team.workitem.WorkItem/_0J39QJu-Ed6cerS9lb5AWw
Accept: application/x-oslc-cm-change-request+xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<oslc_cm:ChangeRequest
  xmlns:rtc_cm="http://jazz.net/xmlns/prod/jazz/rtc/cm/1.0/" xmlns:oslc_disc="http://open-services.net/xmlns/disc
  xmlns:dc="http://purl.org/dc/terms/" xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/"
  xmlns:jp="http://jazz.net/xmlns/prod/jazz/presentation/1.0/" xmlns:jd="http://jazz.net/xmlns/prod/jazz/discover
  xmlns:oslc_cm="http://open-services.net/xmlns/cm/1.0/" xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:calm="http://jazz.net/xmlns/prod/jazz/calm/1.0/">
  <dc:type rdf:resource="https://rtc:9443/jazz/oslc/types/_gasc4Ju-Ed6cerS9lb5AWw/defect"/>
  <dc:identifier>9</dc:identifier>
  <dc:created>2009-09-07T14:59:06.333Z</dc:created>
  <dc:creator rdf:resource="https://rtc:9443/jazz/oslc/users/_6I8ZMJu9Ed6cerS9lb5AWw"/>
  <dc:title>My First Bug</dc:title>
  <dc:description>This is my first bug</dc:description>
  <dc:subject/>
  <dc:modified>2009-09-07T14:59:06.348Z</dc:modified>
  <oslc_cm:priority rdf:resource="https://rtc:9443/jazz/oslc/enumerations/_gasc4Ju-Ed6cerS9lb5AWw/priority/priori
  <oslc_cm:severity rdf:resource="https://rtc:9443/jazz/oslc/enumerations/_gasc4Ju-Ed6cerS9lb5AWw/severity/severi
```

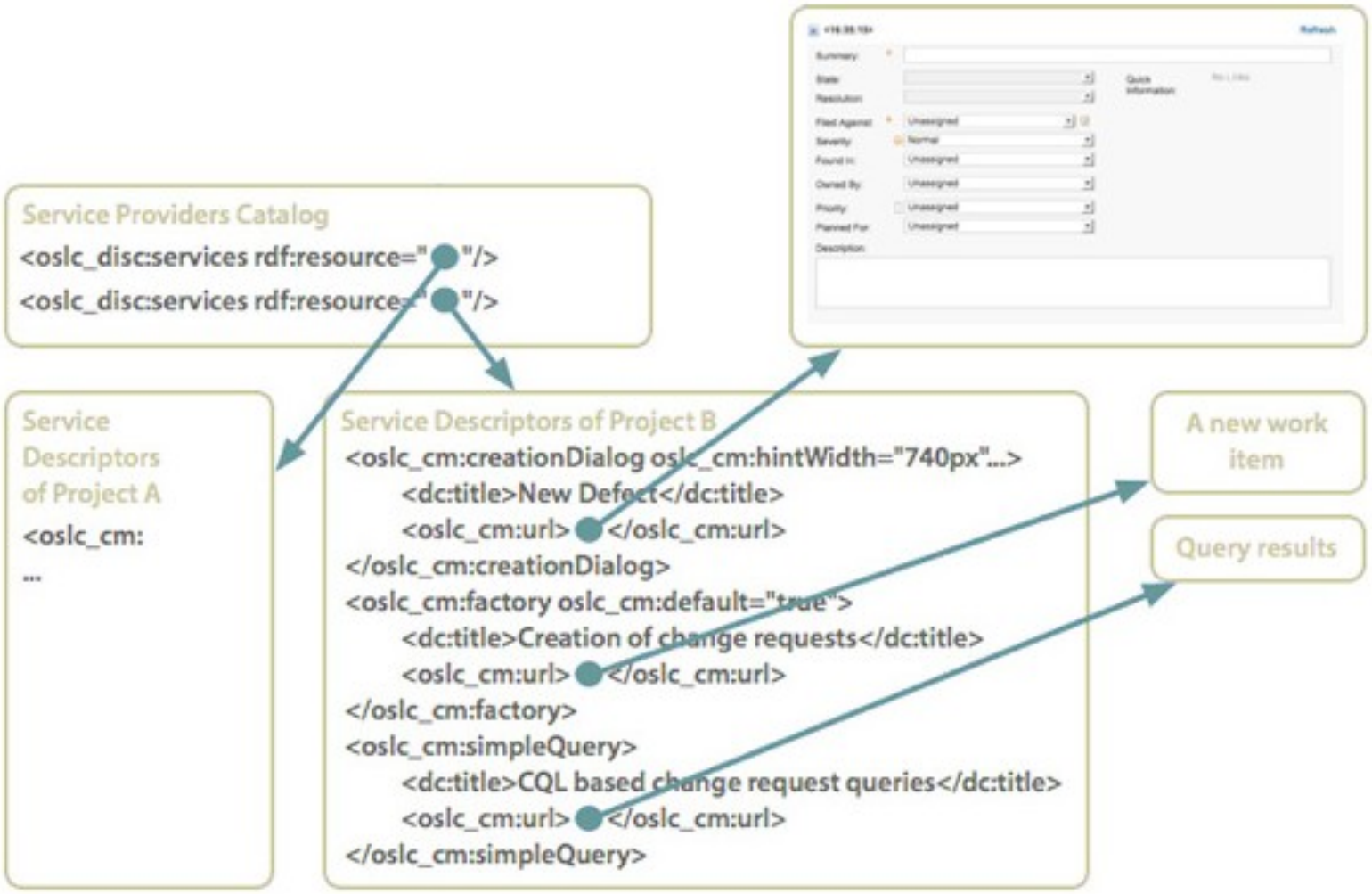
Service Discovery

1. Discover the existence of the Change Management system itself, known URL
   - E.g. https://rtc:9443/rtc/rootservices
1. Discover the contexts (e.g. projects) in which change requests may exist, e.g project
2. Discover the services that are provided within that context

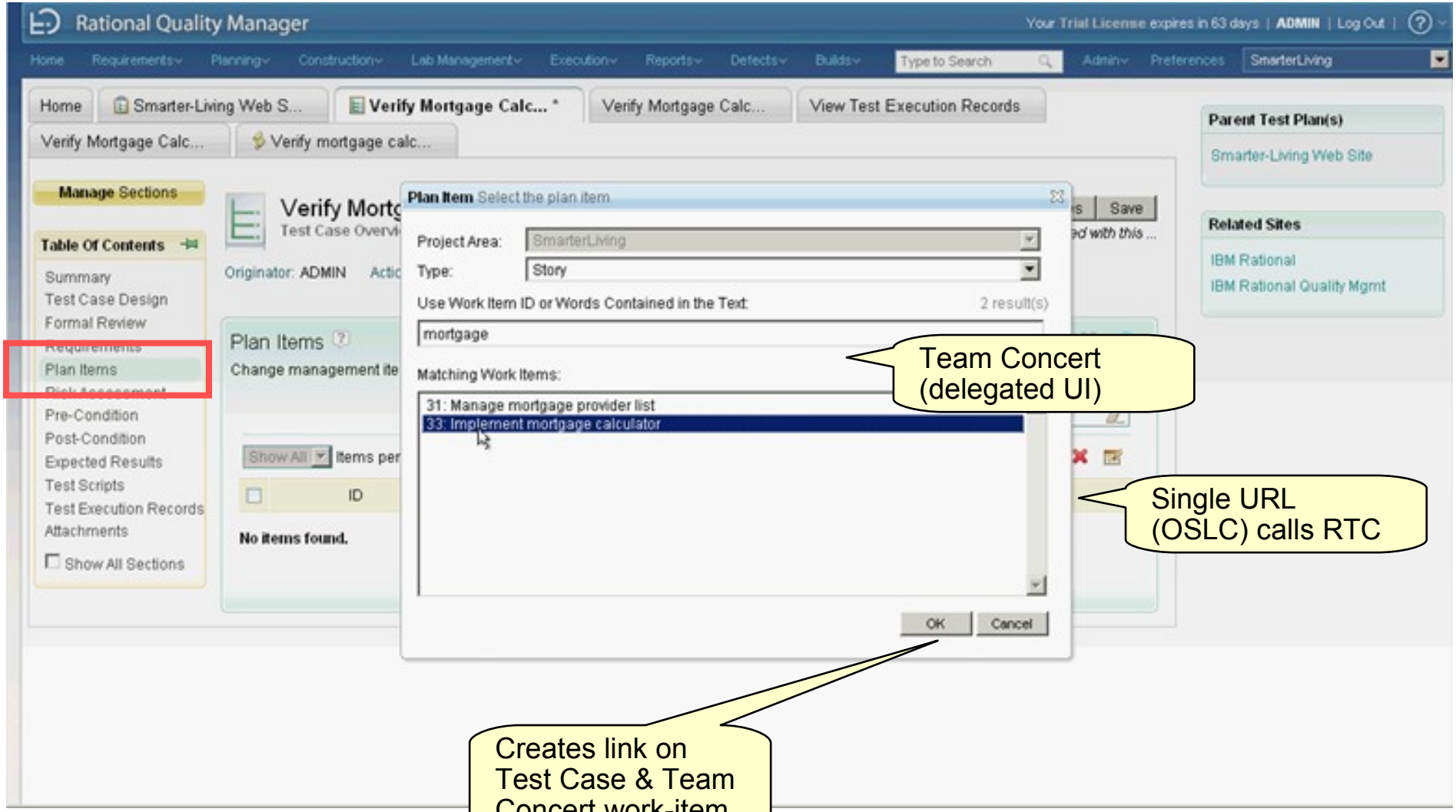



```
<rdf:Description rdf:about="https://rtc:9443/jazz/rootservices">
   ...
   <oslc_cm:cmServiceProviders rdf:resource="https://rtc:9443/jazz/oslc/workitems/catalog"/>
   ...
</rdf:Description>
```

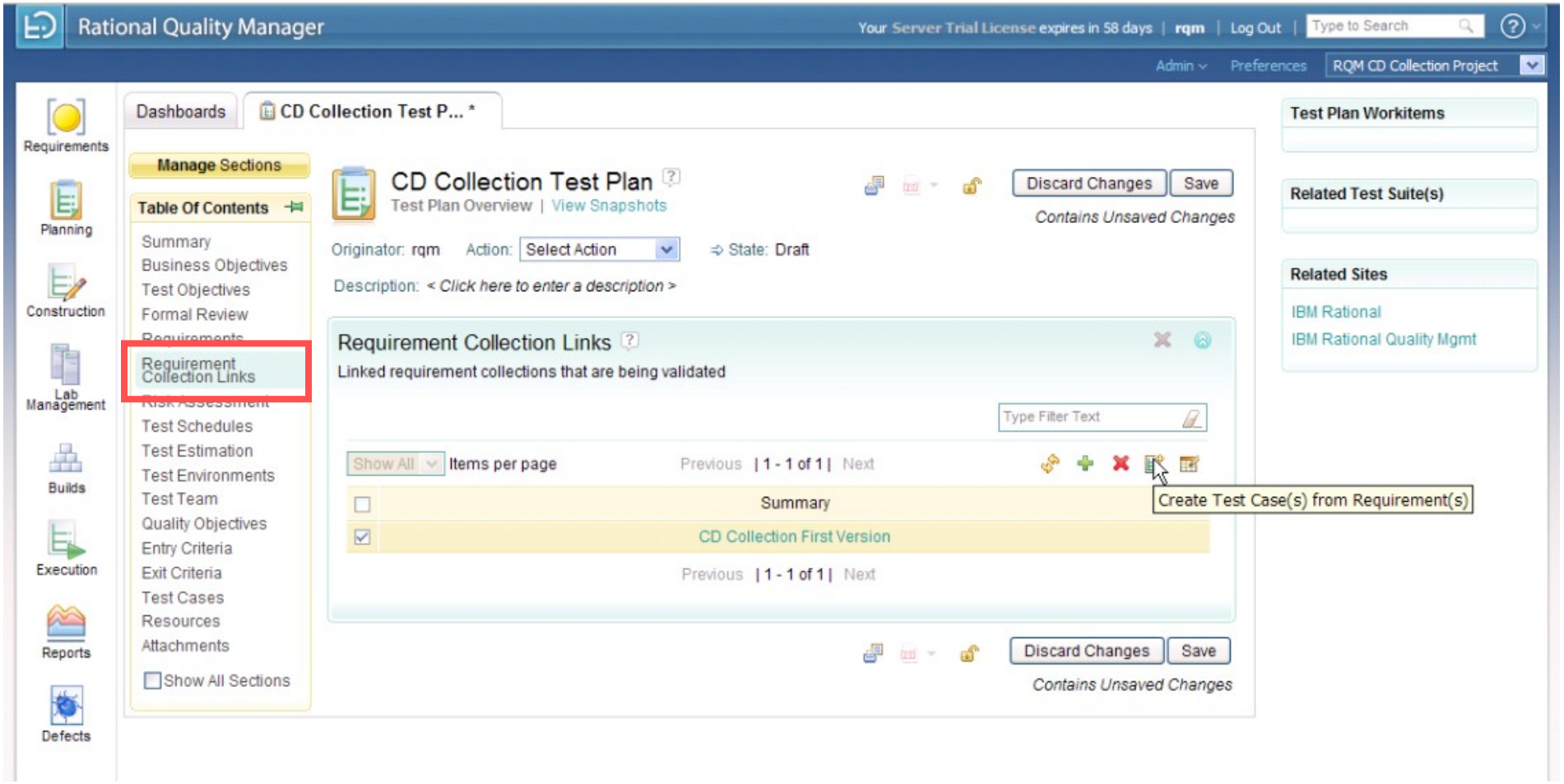
Discovering the Creation Dialog



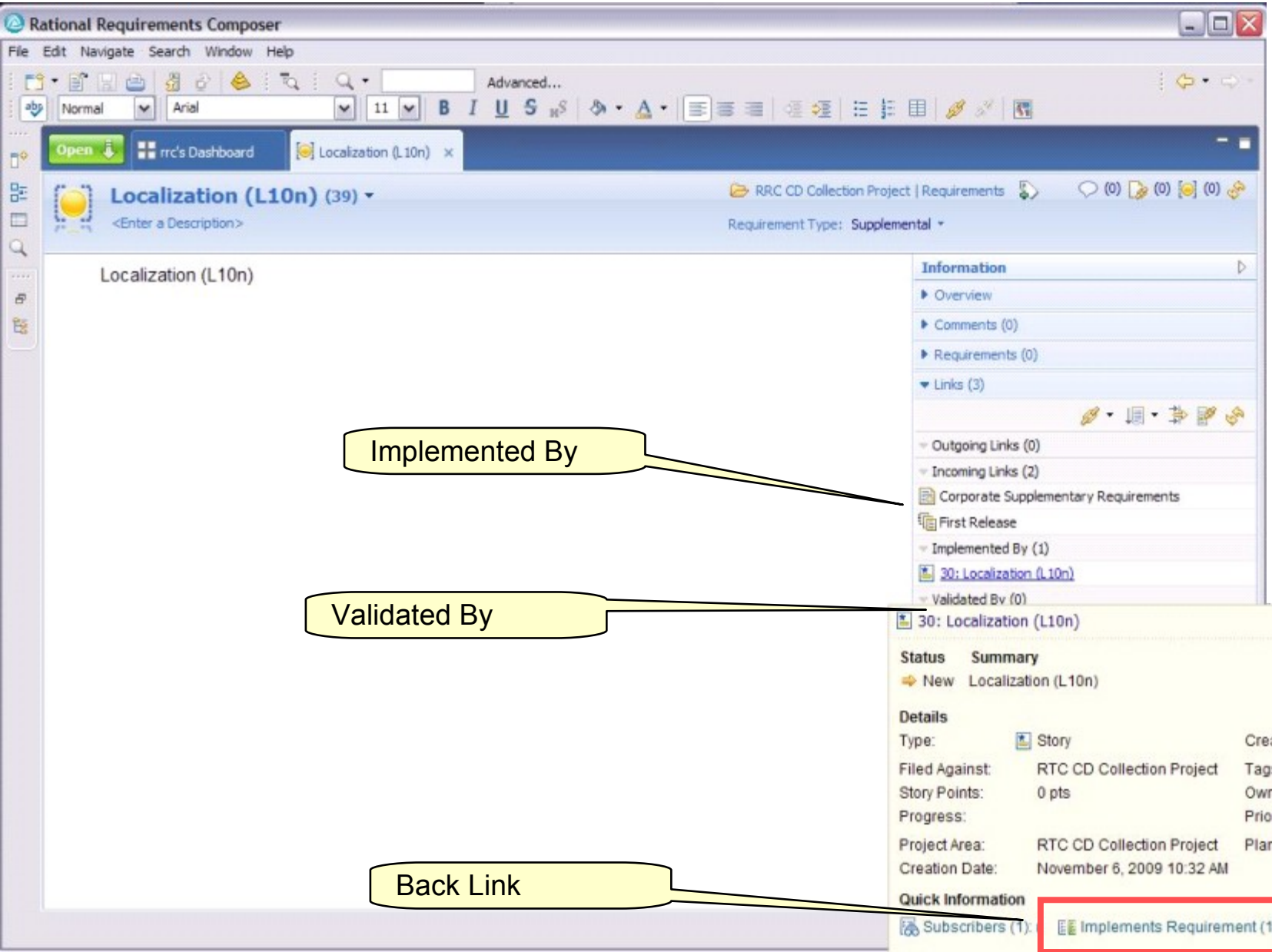
OSLC example: What are you testing?



OSLC example: Creating Test Cases from Requirements



OSLC example: Resource Links in Requirements Tool



What Makes the OSLC Approach Better?

Traditional Approach

- Brittle integrations, version-specific APIs
- Monolithic repository or import/export
- “Boil the ocean” meta-model design
- Forced migration to a common code base
- Premature architectural decisions
- A vendor-led “partners” program

OSLC Approach

- Loosely-coupled
- URLs
- Minimalist
- Technology-neutral
- Incremental
- Open

See it live at Jazz.net

- Transparent development
  - Jazz architecture
  - Jazz products
- Self-hosting
  - Using Jazz products…
  - … to develop Jazz products
- Learn about Jazz at Jazz.net
  - Participate in the evolution
- Try it
  - Sandbox available