IBM Software Group

Shifting Requirements Drive New Architecture - a story

Martin Nally, IBM Fellow, Rational CTO

Rational. software

ON DEMAND BUSINESS

IBM Rational in 2004

- A leading provider of Application Lifecycle Management Tools
  - Source Code Management (ClearCase, #1 market share)
  - Defect/Change Management (ClearQuest, #1 market share)
  - Requirements Management (ReqPro, #1 market share).
  - …
- Mature products
  - Designed and built in the ’90s
  - Evolving to the web

Product Architecture Matches Requirements

- Classic client/server architecture
  - Extensive business logic in clients
    - Responsive UI for desktop users
    - Lots of local desktop validation
    - Extensibility architecture allows logic flexibility and UI tailoring
    - Exploit local processor resources
  - Database on server
    - Exploit high-bandwidth LAN communications
    - Vertical scaling of database servers
    - Central administration teams provide “tools-as-a-service”

End of story?

Times change – requirements change

- New Requirements
  - Distribution of development teams around the world
  - Want more tightly integrated end-to-end ALM solution
  - Want broader solutions
  - Need greater agility and lower cost of ownership
- Response
  - Adapt existing products to the new environment
- Necessary, works, but has limitations

Time for a new architecture

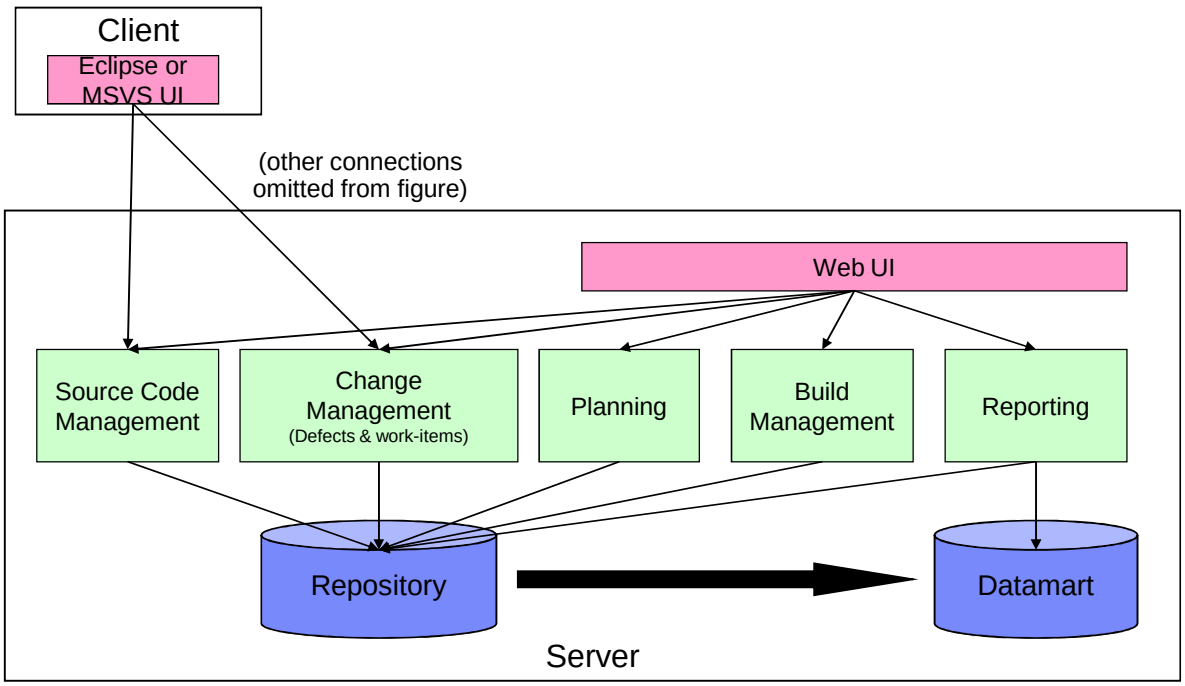
Jazz Phase I

- A new Application Lifecycle Management suite (Rational Team Concert - RTC)
  - Much tighter integration of source-code management, defect/change management, build, planning, reporting
    - Integration of data
  - Modern collaboration tools (Wiki, blog, IM, feeds, presence, …)
    - Integration of people
  - Enactment of process (tailorable)
  - Complements existing IDEs (Eclipse, MSVS, VI/emacs)
  - Very lightweight install, configure, admin
- Ability to add more tools over time
  - Rational Quality Manager (RQM)
  - Rational Requirements Composer (RRC)
  - Others …

Jazz Phase I

- A new architecture and platform for building tools
  - 3-tier architecture
  - Use of standard application-server middleware
  - Use of a standard relational database backend
    - Powerful, flexible user query
  - Web browser as well as Eclipse-based user interfaces (MSVS too)
  - Java as a programming language
  - OSGi as a modularity, packaging and deployment technology
  - Eclipse Modeling Framework (EMF) as a framework for serializing objects passed from client to server and for object to relational mapping
  - Web service interfaces
  - Atom feeds for events and query results

RTC approximate architecture



Jazz Phase I – is this a good architecture?

- Yes!
  - Modern
  - Scalable
  - Proven
  - Open
    - Add OSGi bundles
    - Integrate via HTTP web services
    - Feeds
  - WAN-friendly
    - Web UI in addition to IDE integration
    - IDE integration also uses HTTP

End of story?

New Requirements

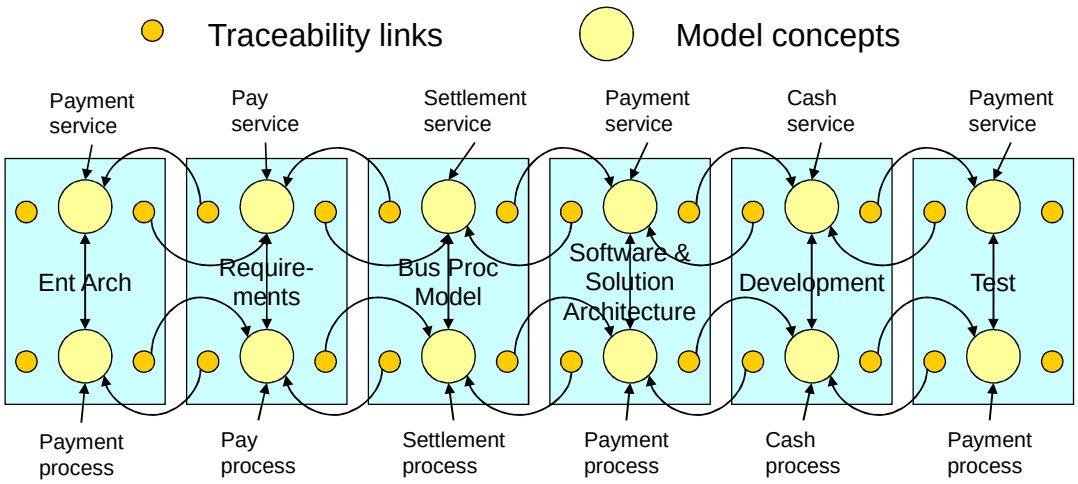
- Scale beyond RTC to many, many tools
- Integrate with a lot of tools and data that already exist
- Integrate across client installations
- Evolve clients independently of servers
- Evolve data model when large data sets exist
- Allow new tools to extend existing data

Time for a new architecture

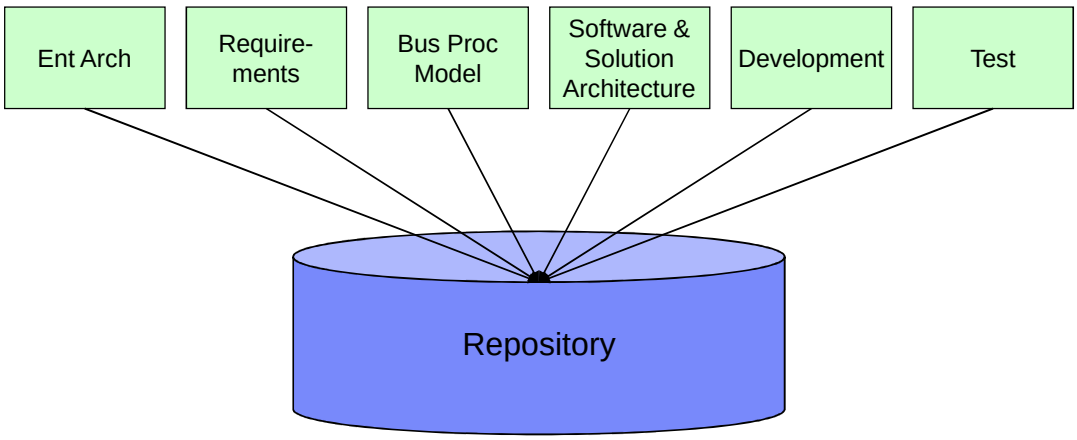
Jazz Phase II – Jazz Integration Architecture (JIA)

- A complement to Jazz phase I, not a replacement
- Integrates across products when
  - You can't get on one database
  - You can't get on one schema
  - You can't get on one delivery schedule
  - You can't impose a tool technology choice
  - You have to integrate existing tools and data

Architecture 0 - Data Integration the old way



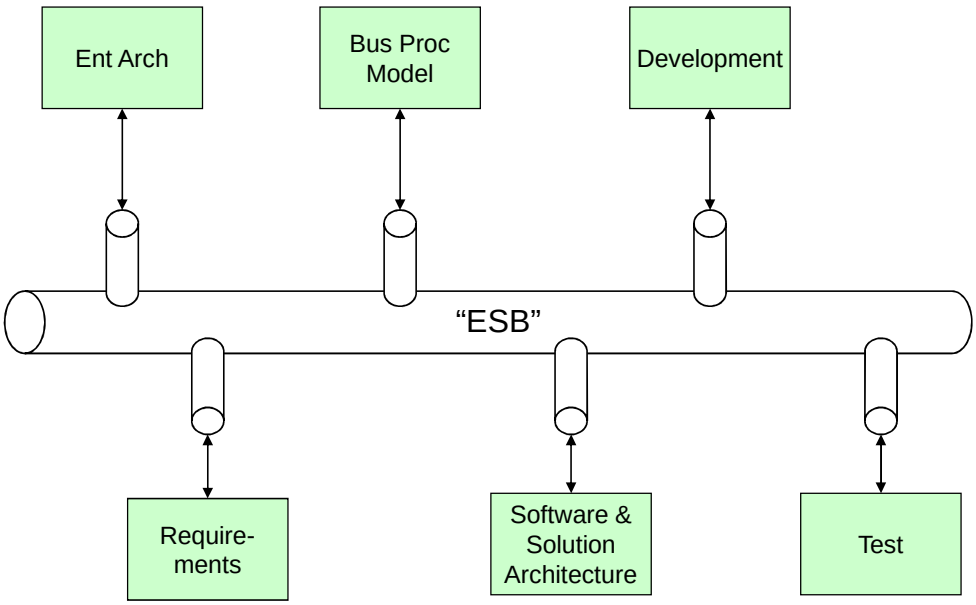
Alternative Architecture - 1



Alternative Architecture - 1

- Tried many times before
  - AD/Cycle, PCTE, …
- Pros:
  - Effective when the function can be delivered and deployed in a highly-coordinated fashion
- Cons:
  - Requires schema coordination across tools
  - Does not provide a mechanism for integrating “existing ” or “foreign” tools
  - Requires “central planning” by consumer (deploy 1 database) as well as provider (integrate tools to a single schema)

Alternative Architecture - 2



Alternative Architecture - 2

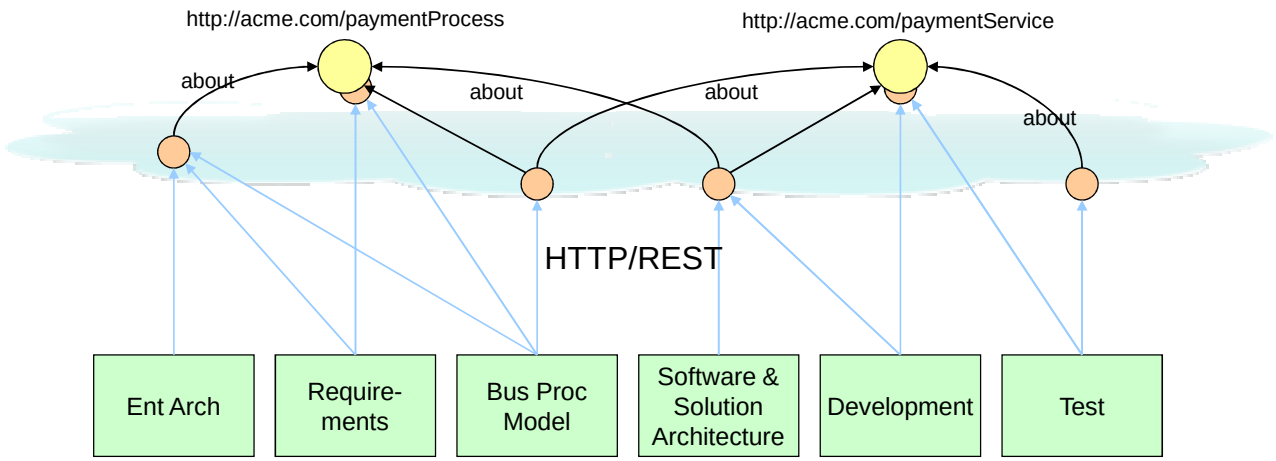
- Pros:
  - Allows integration of existing applications
  - Some of the process implemented in the “bus”
- Cons:
  - Multiple copies of data in different tools
  - Sensitive to changes in data model in any participating tool
  - Sensitive to changes in the process
  - Requires a lot of centralized implementation

Problem

- We've been stuck with these problems and solutions for 20 years
- Is Martin going to claim he's invented a new way to write tools that fixes this?
  - Yes, and No
    - We think there is another way
    - We didn't invent it …

…Al Gore did

Alternative architecture 3 – “www linked open data”



Author software artifacts directly on the web in the way that a Wiki lets your author HTML pages on the net

Alternative Architecture - 3

- Pros:
  - Allows integration of existing applications
  - Supports open, federated data model
  - Does not require data copying
  - Supports independent evolution of products
- Cons:
  - Unproven (for our domain, at least)
  - Big paradigm shift (for us)
  - Lots of invention required
- Somewhat like architecture 0 except
  - universal addressing
  - Separation of data from tool

Jazz Phase II

- Problem – controversy over new architecture
- Solution – Articulate requirements and “architectural principles”
  - Integrating existing tools is as important as building new
    - assume a federated, open, changing data model
    - allow tools to be implemented in any internet-aware programming language or platform
  - Assume products must be on independent release schedules
  - Assume multiple products must access the same data
    - separate the implementation of tools from the definition of and storage of the data
  - access and integrate data where it is, don’t import/export
  - support multiple client technologies

Jazz Phase II - realization

- Data represented as Web Resources accessed via HTTP
- Cross-product query provided by “structured index”
  - Google for structured data
- Integrate independent server products around common “services” (e.g. administration, query, storage) invoked via internet messages
- Support UI integration as well as data integration
- Security via OAuth
- Tags, not folders (still some controversy)
- Relationships/predicates, not collections (still some controversy)
- Resource-centric, not application-centric user interfaces
- Exploit RDF, eschew WebDAV
- Programmable resources (but stay RESTful)

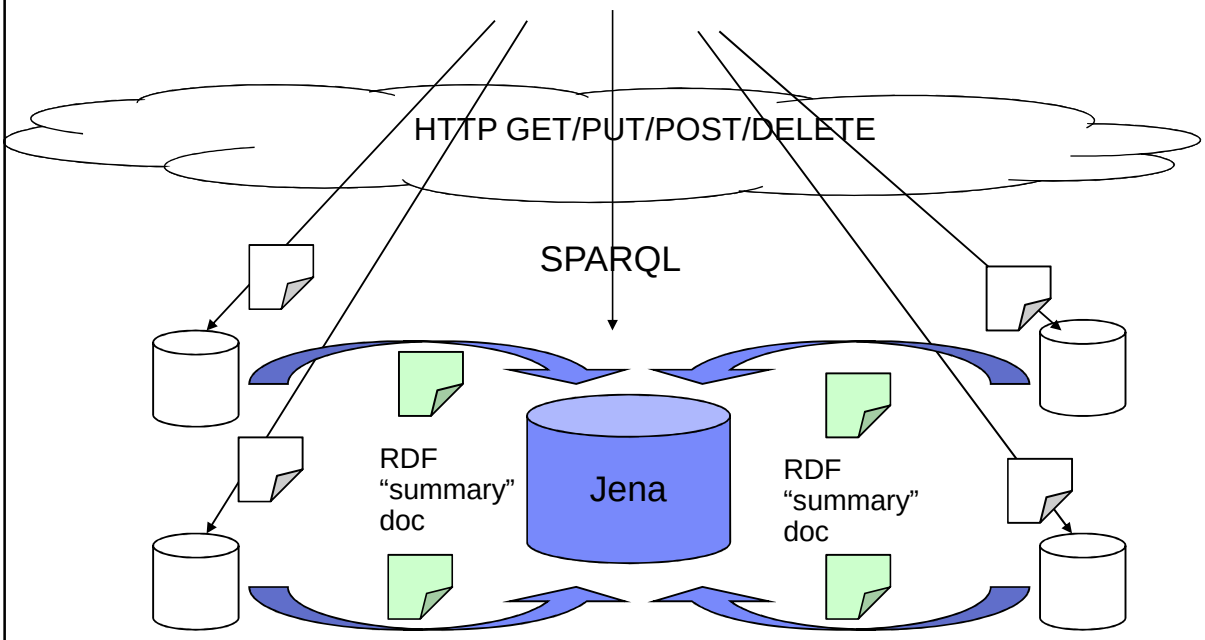
Data represented as Web Resources

- Optional “schema-less” repository implementations with differing QOS
  - Storage for un-interpreted resource representations
  - Non-versioning, linear versioning, branching/merging (e.g. SCM)
- No “internal” product API – all data access is via HTTP
- Representation design is key
  - Simple data
  - Relationships
  - Multi-valued relationships (cf. collections)
- URL design is as important as representation design
- XML is popular, but why?
  - Look at other options.

Cross-product query

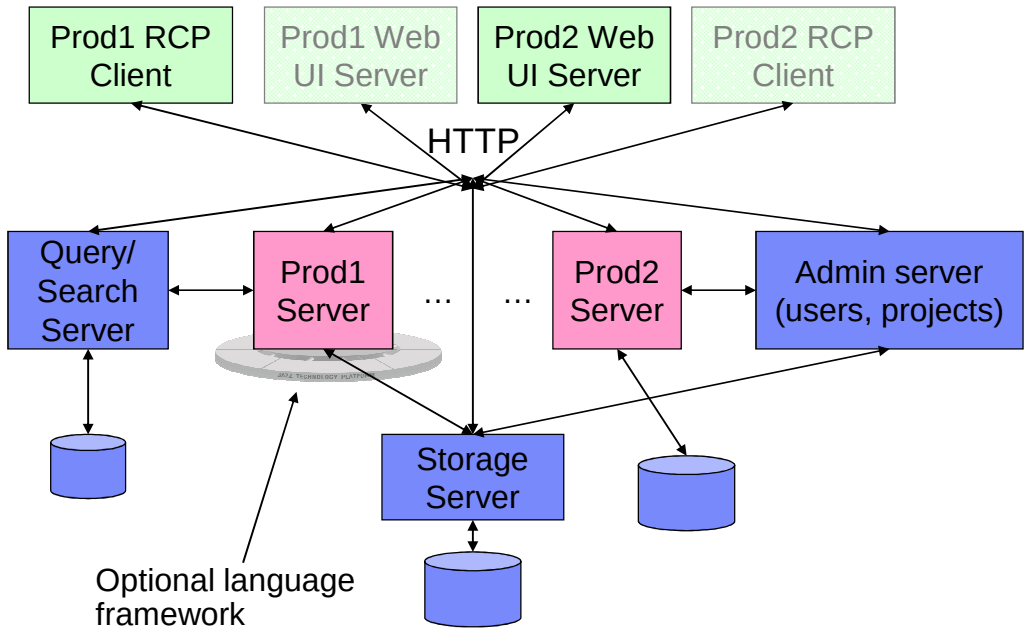
- Query server independent of storage, broader scope
- Schema-less query
- Exploit RDF
  - XML and XQuery were tried, did not work as well

Implementing query on data not in a repository



A federated web architecture with optional frameworks

Note to self – redo this diagram in RSx



example - UI Integration

- Problem:
  - A test product wants to allow testers to open defects in another product
  - Defects are customized – don't know in advance what fields they have
  - Defects have context (projects, components) and "meta-data" (defect states, defect types, …)
- Option 1 – teach test tool about defects, their types, constraints, contexts
- Option 2 – let the test tool use a UI component from the defect tool to create the defect
  - allows much simpler interface, looser coupling
  - Relies on human third-party
- For 2009, we used a simple component "convention" based on iFrames – looking at Open Social Gadgets

Security

- Requirements
  - Allow each product to have its own security administration and authentication
  - Provide optional central place to configure users, integrate with LDAP etc
  - Allow each product to manage it's own access control lists, pre-conditions and post-actions
  - Provide optional central federation of ACL administration
- Solutions
  - For products that implement their own authentication, federate with OAuth
  - For products that want to manage ACLs, provide federation
  - Provide optional building blocks for new-construction that take over admin, authentication and ACLs

How do you write an Ajax UI?

- Attempt 1 – AJAX clients handle multiple-resources
  - Pros:
    - Responsive UI – no page refresh between resources
  - Cons:
    - Complex clients (1990s client-server)
    - Closed system (can't link to unexpected resources)
- Future attempt:
  - Page-per-resource
  - Ajax inside pages

Miscellaneous architecture/design

- Tags, not folders (still some controversy)
- Relationships/predicates, not collections (still some controversy)
- Exploit RDF, eschew WebDAV
- Programmable resources (but stay RESTful)
- Back links for inverse navigation?
- Discovery
- What is userID?
- Transactions

Shifting to a web-centric design

Old	New
Services	REST
Files	Resources
Repositories	Linked open data
J2EE apps	Heterogeneous apps
Tightly-coupled apps	Loosely-coupled apps
Relational/XML data	RDF
SQL/XQuery	SPARQL
WebSphere/Tivoli authentication	OAuth
Folders	Tags
WebDAV	RDF
Collections	RDF Predicates

Jazz Phase II experiences

- Changing paradigm is hard
  - REST is a data-centric design paradigm, quite different paradigm to services
  - Some internal skepticism had to be overcome
  - Simple in concept, many, many issues have to be worked through
  - “Organizational change” is harder than technology change
- Traditional industry middleware offerings not as helpful as hoped
  - Schema-less storage services built on relational (DB2 and competitors)
  - Schema-less query built on open-source (Jena)
  - Application Server provide a useful server base (Tomcat or commercial)
  - Cross-technology security built on open source (OAuth)
- This much architecture shift is high risk
  - You have to believe the benefits will be high to try it
  - Example: 2 years experimentation to get a stable, performing query solution

Complementary architecture – data warehouse