SAVARA and SOA Repository

The Problem

- Software Development Projects involve
  - Many people in different capacities (roles), from high level business sponsers/users, analysis, architects, designers, developers, QA testers, production engineers, etc
  - Many stages in the lifecycle, from business analysis (requirements capture), architecture, service oriented analysis & design, implementation, testing, deployment and monitoring/management
  - Usually multiple artifacts involved at each stage of the lifecycle
  - Systems comprised of multiple services, where each may have multiple dependencies on shared artifacts
  - Geographical distribution of project teams

The Problem (2)

- This leads to many people trying to maintain:
  - a large number of inter-related services and artifacts,
  - across multiple repositories, and
  - ensure they remain valid and consistent
- Apart from the lack of management, this also leads to a visibility problem in respect of the business users
  - business users can only interpret the high level requirements and design documents
  - they have limited visibility of the stages and deliverables of each phase of the lifecycle

The Solution

- A solution to this problem must address the following requirements:
  - Management of services and artifacts (dependencies & lifecycles)
  - Validation of services and artifacts against governance policies
  - Conformance/consistency checking between dependent services/artifacts
  - Tool support for rectifying conformance/consistency issues
  - User/Role based service and artifact browsing
  - Service deployment from consistent and valid artifacts within the repository

Requirement 1: Management of Services and Artifacts

- Federated repository of services and artifacts
  - A federated approach is required as services and artifacts will be stored in different types of repository (e.g. Network file systems, wiki, subversion/cvs, maven, proprietary databases, UDDI, etc.)
- Manage dependencies (links) between services/artifacts
  - In the federated repository, these links need to potentially be managed across different underlying repositories
- Manage service, artifact and group lifecycles
  - Artifacts may need to be managed in logical groups
  - Individual services, artifacts or groups may need to be tracked against phases in a lifecycle

Requirement 1: Management of Services and Artifacts (2)

- Manage service, artifact and group lifecycles
  - Artifacts may need to be managed in logical groups
  - Individual services, artifacts or groups may need to be tracked against phases in a lifecycle
- Authorization based lifecycle change
  - A change to a particular lifecycle phase may require approval from one or more people within an organization
  - Approval request should be accompanied by an impact analysis of the requested lifecycle change
  - Authorization procedure may need to be customizable

Requirement 1: Management of Services and Artifacts (3)

- Support for collaboration
  - Need to track modification history (versions) and enable comparison between versions
  - Enable version history graph to be viewed
  - Comments should be recorded against each stored version
  - Support informal review comments associated with a service and/or artifact
  - Support links between a service/artifact and ticket(s) within an issue tracking system
  - Support user defined tags, to facilitate advanced search capabilities

Requirement 2: Validation of services and artifacts against governance policies

- Tools used in creation of service/artifact may provide validation related to particular syntax or semantics
  - Where this is not the case, a per service/artifact validation capability (with pluggable rules) would ensure that each service/artifact was classified as being valid before it is used in subsequent phases of the development lifecycle
- Organisations may wish to provide additional governance 'rules' for specific service/artifact types
  - Such rules may be related to standards, compliance regulations or corporate policies (e.g. Conformance to WS-I profile, coding standards, test coverage, etc).

Requirement 3: Conformance/consistency checking between dependent services/artifacts

- As services/artifacts are modified, we need to ensure that any dependent services/artifacts are not adversely affected
  - Process Governance can be used to analyse process behavioral differences (for relevant artifact types)
  - "*What-if*" capability should be provided to enable the effect of the change to be explained to the user before they make the change public in the repository
- Suitable notification mechanism must be provided
  - User that has applied the change should be notified of the effects
  - Users responsible for affected services/artifacts should be notified of the impact, along with other people that have registered interest

Requirement 4: Tool support for rectifying conformance/consistency issues

- Identification of conformance issue should determine nature of the differences
- Difference information should be used to help guide modification of affected services or artifacts
  - In some situations the source service/artifact that has been modified will need to be reverted, as it must conform to the target service/artifact
  - In other situations, the target service/artifact must be updated in-line with the new representation of the source service/artifact
- Tool support may be web based, to directly help fix services/artifacts in the repository, or Eclipse IDE based where development artifacts are affected

Requirement 5: User/Role Service and Artifact Browser

- Storing the services and artifacts, and the dependencies between them, is only one part of the problem
  - We need advanced ways to navigate the potentially large amount of inter-related information (services, artifacts, comments, tags, different lifecycles, etc)
  - User/Role specific filters may be required to only show pertinent information
  - Service and Artifact editor/viewers may need to present information in different styles/levels of abstraction, suitable to the user/role

Requirement 6: Service Deployment

- Repository defines dependencies between service and required artifacts
- Step required to package service and relevant artifacts in a deployable unit
  - Within or outside the control of the repository
- Association of deployment unit with IT resources (service containers) capable of executing service
- As packaged service enters 'test' and/or 'production' lifecycle phases, deployment could be automated to associated service containers
  - Immediate or specific time based deployment

Use Case 1: Requirements gathering

- Requirements may initially be defined in a number of unstructured documents in different repositories
- Structured requirements can be defined to support a 'testable' approach
- A federated repository can be used to
  - Define dependencies between the structured requirements, and the more adhoc unstructured documents from which the requirements were derived
  - Provide comments on how the requirement was derived from the source material
  - Enable a reviewer to understand the source of a particular requirement via easy navigation to the original material

Use Case 2: User feedback

- Business users may need to review and comment on artifacts at various stages of the lifecycle
  - Generally will be at the earlier stages, related to requirements or architectural models
- Developer peer review
  - Design and implementation artifacts will need to be reviewed by peers, with their comments being associated with the relevant artifacts
- Feedback can be used as
  - Actions to rectifying short term problems
  - Input to requirements for subsequent phases (as part of a change management process)
  - General comments with no action required

Use Case 3: Historical context

- Systems are developed over long periods of time
- They involve different people at different stages
- They involve different people on the same stage at different times
- Historical version information and comments, along with dependencies between relevant artifacts, can help maintain an understanding of why certain decisions were taken

Use Case 4: Change Management

- Maintaining of dependencies between components is important when we need to make changes to an existing system
- When a change has been identified, the dependency information can be used to determine whether the change has an adverse effect on other services or artifacts within the repository
- Knowledge of the impact of a change can enable careful planning to avoid issues when the change is deployed into a production environment