SAVARA

Project Charter

Version 1.0

Date: 18th August 2009

Table of Contents

1 Introduction 4
2 Testable Architecture Methodology 5
2.1 Architecture Specification 7
2.1.1 Requirements: Defining Communication based Scenarios with Example Messages 8
2.1.2 Global Model/Choreography 9
2.1.3 Message Schema Definition.....10
2.1.4 Outline Deployment Model 10
2.2 Service Specification 10
2.2.1 Local Model 10
2.2.2 Service Level Agreement 11
2.3 Service Development 11
2.3.1 Service Design 12
2.3.2 Data Model Design 13
2.3.3 Service Implementation 13
2.3.4 Detailed Deployment Model 13
2.4 Testing 14
2.4.1 Component Unit Testing 14
2.4.2 System Integration Testing 15
2.5 Documentation 16
2.6 Deployment 16

2.7 Runtime Monitoring 16
3 Tool Architecture 17
3.1 Repository 17
3.2 Managing Users and Tasks 18
3.3 User Interface - Navigating, Creating and Editing Artifacts 18
4 Project Governanace.....19
4.1 Aims.....19
4.2 Project Board.....19
4.3 Working Groups.....20
4.3.1 Methodology.....20
4.3.2 Compliance and Standards.....20
4.3.3 Tooling.....21
5 List of Contributors.....21

1 Introduction

SAVARA is a new community project established by JBoss/RedHat, in collaboration with Cognizant Technology Solutions, to provide a framework for Enterprise and Solution Architects, based around a new methodology called "Testable Architecture", used to build distributed systems of which service oriented systems are an embodiment.

The difference between the tools that will be developed as part of this project, and other enterprise architecture tool suites, is that the goal of this project is to ensure that all artifacts created throughout the lifecycle of a software development project are verifiable against other previously defined artifacts. Using this approach, it will be possible to ensure that the delivered system conforms to the original business requirements.

SAVARA will build upon the Process Governance capabilities in Project Overlord to ensure that models defined at various stages in the development lifecycle conform to models from the preceding stage of the lifecycle. Runtime Process Governance will also be used to ensure that the running system continues to conform to the original design and therefore requirements.

Although this document will present the "Testable Architecture" as a top down approach, the methodology should also support bottom up and iterative development approaches. It is also not mandatory that the methodology be used from a global model downwards - a user can start using the methodology from a later phase. The only requirement is that artifacts developed in subsequent phases should be verifiable back to the artifacts associated with the first phase used.

We will also investigate techniques to enable the artifacts from preceding phases to be "reverse engineered". For example, a common scenario will be the need to leverage legacy services in new systems being developed. Therefore, if a service design is not available, then techniques could be used to derive the design from the implementation. It should also be possible to reverse engineer a Global Model from multiple interacting Local Models. Some of these areas are research topics currently being explored by our academic partners.

The document is divided into two main sections, the first discussing the "Testable Architecture" methodology describing the proposed phases, and the second discussing the proposed tool architecture.

2 Testable Architecture Methodology

This section outlines the "Testable Architecture" methodology, based around the emerging BPMN2 standard. In general terms, a "Testable Architecture" can be thought of as any capability that enables use cases to be specified that can subsequently be used to validate/test a model and that model can be used to drive delivery.

Although BPMN2 will provide the core models used in specifying the architecture and resulting service behaviour, other methodologies (and models) will be used where appropriate to provide useful tools for use by Enterprise and Solution Architects.

The principle focus of the methodology is on building communication oriented systems, that is systems that are distributed in nature and achieve their business goals through interaction. This does not mean that data (or information) modelling in an organisation is less important, but in terms of this methodology, communication is the primary concept. For example if we have a solutions architect, responsible for the global model, and a data architect, responsible for the data/information model, there maybe iterative cycles between the two in which the solutions architect provides requirements concerning identity over conversations to the data architect and in which the data architect provides pre and post-condition as requirements to the solutions architect on the interactions that underpin the global model.

As will be discussed in the relevant sections, corporate information may be used:

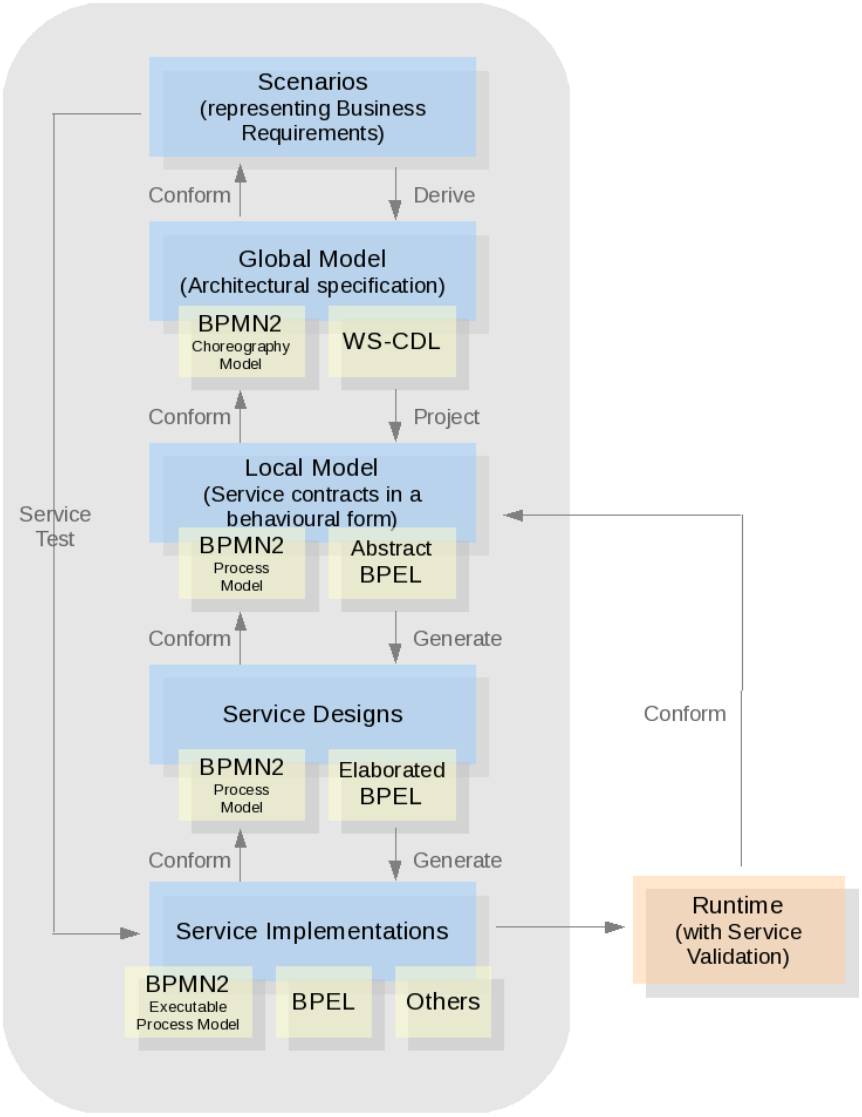
(1) by a single service - where the service is providing an 'added value' interface to the underlying information

(2) by multiple services - where each client service will have specific queries that need to be performed. These can be modelled in terms of interactions on a logical service that represents the information model.

By dealing with data/information models in this way, it is possible to understand what information is required to support the clients of those information sources, and then define the underlying information models to meet those needs as part of the 'service development' phase.

The following diagram represents a high level view of the "Testable Architecture" methodology as applied to the interaction/communication oriented view of a system. As previously mentioned, the principal aim of this methodology is to ensure that each stage in the development lifecycle can be verified against the preceding stage. As the diagram

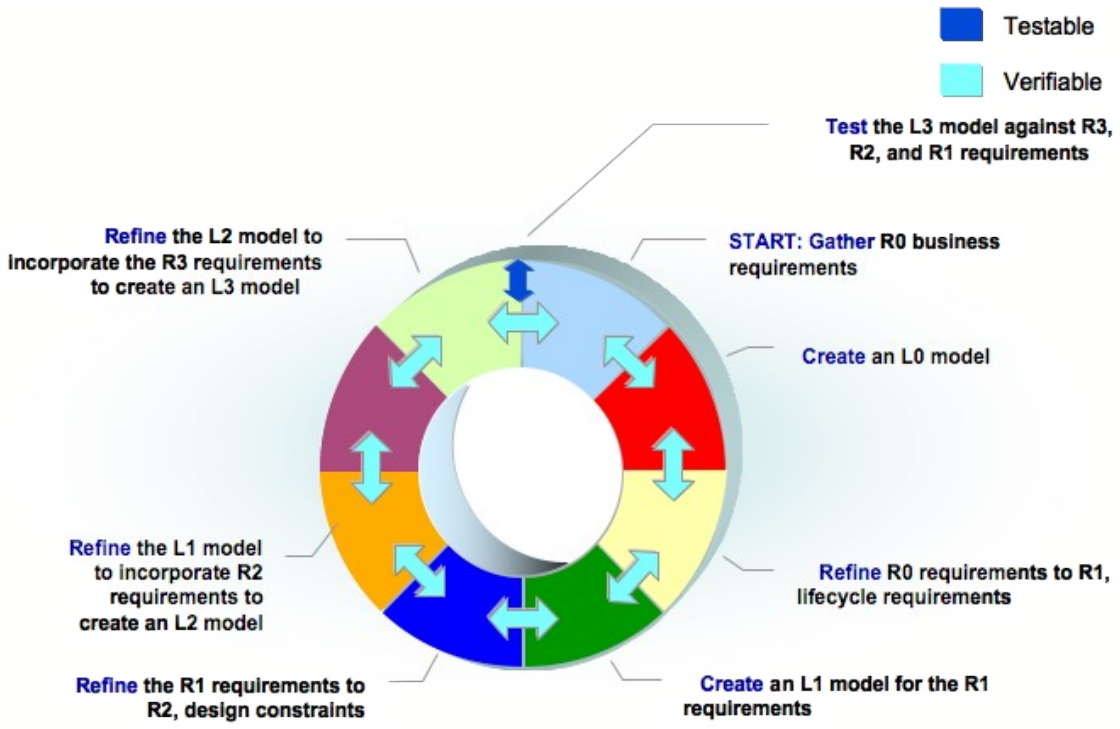
also shows, it is possible to use artifacts from a preceding stage to generate skeleton artifacts for the subsequent stage.



2.1 Architecture Specification

This section outlines the tasks that are performed to define the overall architecture of a system. The steps within this section can be defined in an iterative manner, defining the requirements and associated model in progressive levels of detail until the specification is complete enough to be used to identify the service specifications.

For example, the requirements and associated model could be refined in the following way:



Successive refinement of requirements and models leading to streamlined delivery

This diagram outlines the evolution of business requirements and associated models through a series of levels, where each subsequent level is a refinement of the preceding level, and can be verified against it. Once the requirements and model have reached a suitable level of completeness (level 3 in this diagram), then the requirements can be tested against the model.

2.1.1 Requirements: Defining Communication based Scenarios with Example Messages

Scenarios represent the interaction based use-cases, to describe how the various components in a system will interact to achieve certain business goals. Participants in these scenarios may represent software components or people (human roles). Interactions with people can be achieved through workflow/task management (such as WS-HumanTask).

Scenarios can be represented using a simplified form of UML sequence diagrams, with example messages attached to each interaction. Assertions can also be defined, to indicate conditions that must be met by messages generated by services. Each scenario represents a particular path through the business process being developed.

In terms of the "testable architecture" methodology, these scenarios represent the high level business requirements for the system. Therefore ultimately it must be shown that each scenario has been satisfied by the implemented system.

The vertical lines in the scenarios represent roles being enacted. This does not necessarily mean that each role equates to a service. It may be that a service will implement multiple roles. When the scenarios are initially defined, it should be based on a logical separation of responsibilities. When defining the choreography, it may be necessary to refactor the roles, and possibly this is something that should be supported by the tooling.

The scenario can be specified by defining the interactions that occur between each of the roles. The interactions will define the necessary message type details, to distinguish them from other interactions, and to provide a business context to the message exchange.

As part of the scenario, the user will be required to define example messages. Although the actual message implementations may be defined in a variety of formats, from a specification perspective we will define message content in a neutral XML format. This can be transformed into an appropriate implementation format for testing purposes.

The association between example messages, and the interactions they relate to within a scenario, will be represented in such a way to enable the scenario to be reused against different sets of example messages.

Unlike standard UML sequence diagrams, it will be possible to express a timeline over which the scenario occurs. For example, it will be possible to define 'time compression' to enable the scenario to simulate a significant lapse in time, which may result in a timeout action being taken within affected services.

As well as scenarios being used to defined valid paths through a business process, it will also be possible to define invalid paths. These represent negative tests that ensure the system does not permit invalid use-cases to occur.

2.1.2 Global Model/Choreography

The Choreography Model provides a global perspective over the interactions that can occur between services in an architecture. It defines the dynamic "behavioural type" of the architecture. It provides the 'type' definition that encompasses and aggregates the various paths expressed within the individual scenarios. As such, it will only be considered valid, and therefore meeting the overall business requirements, when it can successfully be verified against the previously defined scenarios.

Where a scenario has been defined to express an invalid path through the business process, validation of this scenario against the choreography should correctly highlight the invalid interaction(s), to demonstrate that the choreography model does not inadvertantly support invalid paths.

The Scenario editor should provide support for manual simulation of a scenario against a choreography model. The results should be overlaid on top of the scenario notation, showing successful interactions in green and failed in red. However continuous validation of the scenarios against the choreography model should be performed when either changes, reflecting any validation errors against the choreography and scenario.

The Choreography (Global) Model can be used to derive a Local Model per participant in the choreography. The use of the Local Model will be described in a following section.

The Choreography Model should be extensible to enable assertions, constraints and policies (e.g. SLAs) to be defined. Assertions and constraints may be used to indicate aspects of messages that must be consistent within a business transaction. SLAs defined within a global model can be used to specify quality of service metrics that span a scope wider than an individual component. For example, a critical SLA may relate to the time it takes for a transaction to initiate a particular interaction with one service, and for another interaction to occur between two other unrelated components.

The assertions/constraints and policies defined in the global model could be monitored and enforced as part of the runtime montoring mechanism.

2.1.3 Message Schema Definition

The Global Model or Choreography provides the dynamic behavioural type that represents the scenarios, however it does not define the static types associated with the messages being exchanged between the communicating services.

This is achieved by deriving (or re-using) a schema that can accomodate the message content as defined in the example messages associated with the scenarios.

2.1.4 Outline Deployment Model

The deployment model is an optional part of the "Testable Architecture" methdology that can provide a physical context for the components associated with a system.

Linking the logical service components with the physical deployment can help with project planning and costing, as well as providing the information required to actually deploy the fully implemented system into a test and/or production environment.

The deployment model can represent real or virtual resources.

If a deployment model is defined for an architecture, then validation can be provided to ensure all the participants in the global model are associated with a component in the deployment model. The type of associated component in the deployment model, can also provide contextual information that can help with the implementation of the global model participant (i.e. service, human task management, database, etc).

This phase will define an 'outline' deployment model, as at this stage the decision regarding the actual deployment technology/platforms may not have been made.

2.2 Service Specification

2.2.1 Local Model

The Local Model represents the abstract behavioural interface of the service component. This model is used to provide a simple definition of the behaviour required to use the

service, as well as the behaviour the service expects of its partners.

Where a Global Model has been defined, it is possible that the Local Model does not need to be explicitly persisted. It can be used in a transient manner, derived from the Global Model, and used where appropriate to either generate the skeleton for the Service Design artifacts, or be used to check conformance of the Service Design artifacts whenever they change.

One place where this model may be persisted is as metadata associated with the service implementations within a Service Registry/Repository. This can then be used to support behaviour based service lookups.

If the Local Model is persisted within a project workspace, then it will be a stable interface against which the Global Model(s) and Service Implementation(s) must conform.

Therefore, when developing a component that requires the use of a service, the Local Model for that service can be used to develop against, without having to reference service design or implementation artifacts for that used service. At runtime, an appropriate implementation can then be located that implements the Local Model behaviour.

2.2.2 Service Level Agreement

The abstract behavioural specification of a service, as represented by the Local Model, may be accompanied by policies that define its contractual obligations in terms of availability and performance characteristics.

Service level agreements may be tailored to user groups, so a range of policies based on the authentication of the 'user' may be defined.

As with the Local Model behavioural description, associated Service Level Agreements may also be recorded in the repository, for enforcement at runtime.

2.3 Service Development

This section discusses the areas related to service development.

One of the benefits of the "Testable Architecture" approach is that it enables different

aspects of a system to be built by different groups. Due to the verifiability of different components and phases against preceding phases, and ultimately the originating business requirements, the responsibilities of each implementing group can be clearly defined.

For example, if each service involved in an architecture is being designed/implemented by a separate team, potentially geographically distributed, then each team can be given the Service Specification (i.e. Local Model and optionally a set of SLAs), representing the behavioural contract they must adhere to, and the scenarios that can be used to test the individual service against the original business requirements.

It ensures each service can be independently developed, while still ensuring that when the service components are brought together for integration testing, they will work as required.

At the commencement of this phase, the only technology decision that needs to be made relates to the service interface. This involves the communications technology that will be used to interact with the service (and for it to interact with other dependent services), as well as the message format (e.g. XML, Java objects, etc). Where a deployment model has been defined, this information can be specified against the relevant components.

2.3.1 Service Design

The Service Design represents the elaboration of a Local Model to include the relevant implementation details. The Local Model can be used to generate an initial skeleton version of the Service Design (as a BPMN2 Process Model).

The Local Model defined in the previous phase, which may be transiently derived from a Choreography Model, can be used to perform continuous (or on demand) validation to ensure the Service Design continues to meet its obligations with respect to the Choreography (Global) Model.

The Service Design should ideally provide extensibility to enable additional implementation technology specific information to be captured as part of the Service Design. Where such additional information can be defined, it should also be possible to extend the model validation to enable the information to be validated.

2.3.2 Data Model Design

One way to view a database is in terms of providing a service. It is a shared component that in most cases will be used by more than one service. Even if not shared, it is useful to be able to separate out the persistent data management aspects from the behaviour of a service.

This enables queries to the database to be represented as an interaction in a Scenario, allowing the access to the database to be tested against the business requirements.

Where a deployment model is defined, the 'database' service can be classified based on the type of component used in the deployment model (i.e. a database).

Tools should be provided to enable the database schema to be defined. Where appropriate, existing schema should be used, and if necessary database virtualisation can be used to consolidate various data sources and present in a simple relationship table format (see http://www.jboss.org/teiid).

2.3.3 Service Implementation

The Service Design should be used to generate initial skeleton artifacts for the selected implementation approach/technology.

Where possible, the behaviour should be derived from the implementation to allow it to be checked for conformance against the design and Local Model. If this is not possible, then runtime behavioural monitoring can be used to check the behaviour of the service against the Local Model.

The initial target implementation language will be BPEL, although direct execution of the BPMN2 process model will also be explored.

Subsequent implementation targets may include SCA and other techniques suitable for execution on an ESB.

2.3.4 Detailed Deployment Model

This phase will enhance the previous (optionally) defined 'outline' deployment model, to provide additional technology specific deployment information related to the individual

components.

For example, where the component represents a database, the deployment model may be elaborated to define the type of database being used. Where the component is a service, the deployment model may define an application server and particular technology stack that will be used.

In situations where the link between the component and the associated element in the deployment model has already been defined in a previous phase, the tool support around generating a skeleton implementation of a service or database schema may capture additional information that can be used to automatically enhance the deployment model.

2.4 Testing

There are two types of testing that we will initially be interested in, namely Component Unit Testing and Integration Testing.

2.4.1 Component Unit Testing

This section is named "Component" unit testing, as opposed to "Service" unit testing, because the global model may define components other than just services. For example, some of the participants within a global model may represent a database or a human interface.

As a component within the global model, it will have clear interaction based behavioural boundaries with other associated components. These relationships, and the specific use cases and example messages that are provided in the scenarios, can be used to test a component in isolation.

The output from a component can also be compared against the scenarios used to test the component, to compare the results against the expected messages.

In some cases, the complete message can be directly compared against the expected message, to determine whether the component responded in a valid manner. However, in many cases, the content of the response message type may contain some variable data that does not actually invalidate the test. For example, some message may carry a date/time field related to when it was processed. This would not compare correctly with any example message stored with the scenario. Similarly the ordering of some XML

elements may not be fixed, and therefore the component response may not exactly match the ordering of elements in the scenario example message.

Therefore a message validation mechanism will be required that can flexibly enable the scenario designer to indicate what constitutes a valid response, based on an example message that has been provided. This could be:

1) Precise message comparison
2) Use of message schema to understand ordering issues
3) Inclusion/Exclusion xpath expressions to indicate which parts of the documents should be compared or ignored

The global model may additionally provide constraints that must be satisfied between messages associated with different interactions. In cases where those constraints relate to messages inbound and outbound for a particular component, the constraints could be validated as part of the component's unit test.

2.4.2 System Integration Testing

Although component unit testing can ensure that each component performs as expected against the use cases defined as scenarios, prior to going into production, all of the components of the system will need to be tested as a complete system.

If a deployment model has been specified, and elaborated as part of the service development phase, then it can be used to help automate the deployment of the system components into a test environment that mirrors the production environment.

Although the scenarios could be used to initiate tests across the complete system, at this stage it may be advisable to use an independent set of use cases.

The Integration Tests will be validated using the runtime monitoring mechanism (part of Project Overlord - Process Governance) that will validate the observed interactions between the components being tested against the global model. This test is therefore ensuring that the system as a whole conforms to the global model.

2.5 Documentation

One of the key motivations for adopting the "Testable Architecture" is to ensure that all information captured, from the initial requirements defined as scenarios, through to the service design and implementation, are verifiable and therefore are guaranteed to meet the original requirements, but just as important, remain up to date. If any changes are made at any stage, that are not internally consistent with artifacts defined in other phases, then this is detected so that it can be fixed.

This approach overcomes the common problem in building large scale systems - namely documenting the requirements and design, and ensuring that they remain up-to-date and of value. If the validity of requirements and design artifacts cannot be guaranteed, then maintainence and change management of the system becomes error prone, time consuming and therefore costly.

Although the benefits of an internally consistent and verifiable software development lifecycle are a significant benefit, it does not mean that documentation is redundant. It simply means that paper based documentation no longer becomes the 'master' copy in terms of architecture and design - however it can be useful to promote understanding of how a system operates.

Therefore, to gain full benefit from the artifacts that are collected through the various phases of the "Testable Architecture" methodology, the project will need to provide a framework that can produce custom documentation using the various artifacts as input.

2.6 Deployment

This phase will be similar to the Testing phase. If the optional Deployment Model has been defined, then it can be used in conjunction with the implemented components to deploy the system to a production environment.

This stage will require extensibility to support a wide range of deployment environments.

2.7 Runtime Monitoring

The final stage in the "Testable Architecture" methodology is to monitor the running system in the production environment to ensure that it continues to conform to the

expected behaviour as defined in the Choreography (Global) Model, and each component specifically against their Local Model representation.

Where assertions/constraints and/or SLAs have been defined, whether associated with the Global or Local Models, these can be evaluated by the runtime monitoring mechanism, and any violations reported to the appropriate destination.

3 Tool Architecture

This section discusses the proposed tool architecture to support the project. Although JBoss open source projects are Java based, the tooling architecture will also aim to support Microsoft based organisations.

3.1 Repository

The tooling for the project will revolve around a central repository, used to organise and version the various artifacts that may be created for a system.

The repository will be based on the JBoss Guvnor project, which is built upon the Java Content Repository (JCR) standard specification. Although this is a Java based API, Guvnor will also provide a WebDAV interface to the repository. This will enable Microsoft based tools to retrieve artifacts and submit changes.

The repository will support a validation framework, to verify the artifacts when changes are made.

Guvnor will provide dependency management, to enable relationships between artifacts to be represented. Therefore, when an artifact is modified, other artifacts that are dependent upon the changed artifact can also be re-validated.

Guvnor will also manage the lifecycle of artifacts (and groups of artifacts), using a configurable workflow based mechanism to implement required authorisation procedures.

Notification of relevant changes (and possibly validation results) will be available via Atom feeds.

3.2 Managing Users and Tasks

Users, whether business analysts, solution architects, data architects, service designers or implementers, will collaborate based on a Task Management capability that will ensure users are informed of their responsibilities, and provide the necessary input at the appropriate time.

The tasks may be procedural, reviewing artifacts and approving (or raising issues to be rectified), or instructional, creating new artifacts and specifying the location of the associated artifacts that can be used to perform the task.

Some tasks may be collaborative, where multiple users (possible distributed in many geographical locations) may be working on the same artifacts.

Task notifications may also be automatically created when validation errors are detected between artifacts in different stages of the "Testable Architecture" methodology. This may be tied to the lifecycle phases associated with the artifact. For example, if the lifecycle indicates the artifact is "in development", then validation errors affecting artifacts in subsequent phases may be surpressed. The only validation errors that are relevant during development are those associated with validating the artifact against preceding phases of the lifecycle. Only when the artifact itself is considered complete, and valid with respect to the artifacts in the preceding phases, will its lifecycle be allowed to progress to a 'stable' state. Once this has occurred, validation errors that may now occur between that artifact and existing artifacts in subsequent phases of the methodolgy may be distributed, and used to create tasks that will inform the owners of those artifacts that work is required to bring them up-to-date with respect to an artifact on which they are dependent.

3.3 User Interface - Navigating, Creating and Editing Artifacts

Guvnor will provide a GWT web based user interface, to enable system and service groups to be established, and the various artifacts to be created and managed.

In some cases, a web based editor may be provided to enable users to directly modify artifacts. In other cases, or as well as a web based editor, the WebDAV interface to the Guvnor repository will enable the artifacts to be retrieved into a user's local file system (or IDE) and edited using appropriate locally installed editors.

The models used in the project will aim to leverage standards where possible, making use of third party editors as easy as possible. However in some cases, the extensible nature of even standard models is not necessarily supported in many editors.

Therefore the project will also aim to provide suitable custom editors that can focus on the needs of the particular user type (e.g. business analyst, service designer etc), to ensure that the relevant aspects of the model are easy to define, and appropriate extensions are easily supported.

4 Project Governanace

4.1 Aims

During the early stages of the project, the intension is to minimize the amount of project governance required, so that the overhead of managing the project does not detract resources from actually delivering on its goals.

The following sections are intended to provide guidelines on how the governance of the project may evolve as more corporate members and individuals join the project.

Generally open source projects at RedHat are managed on a very informal basis. The projects comprise of individual contributors that collaborate on code development through a subversion repository, discuss issues via forums, and produce documentation on a wiki.

However, this project is expected to gain the involvement of corporate, as well as individual members, and therefore needs to be managed on a slightly more formal basis. The aim will be to adapt the governance of the project to meet the needs of the project membership as it grows.

4.2 Project Board

The remit of the project board is:

- To set the high level objectives/goals for the project
- To manage the relationship with corporate members of the project
- To oversee the governance of the project

At this stage, there are no guidelines or rules associated with who can be on the project board, how often it convenes, how voting is handled, etc. These items will be addressed as and when the project board needs to adopt a more formal structure.

4.3 Working Groups

This section describes the initial set of working groups that will be established for the project.

It is proposed that each working group will have a lead member. The lead member will decide how best to collaborate with the members of the working group, whether based on regular conference calls or simply using the forums. The collaboration methods can then evolve as the project grows, based on the needs of the individual working groups.

4.3.1 Methodology

The methodology working group will be responsible for defining the «out of the box» methodology that a user can use when initially downloading the tools.

It is anticipated that user organisations and system integrators may want to customise this methodology to meet their own specific requirements, but this initial methodolgy will enable users new to the project to understand how the tools can be used in the context of a methodology that can be used to deliver a «testable architecture».

4.3.2 Compliance and Standards

This working group will be responsible for defining areas of extensibility required in the tooling, and the criteria that must be met by user organisations and system integrators to be compliant with the project.

This working group will also be responsible for liasing with relevant standards groups to ensure that the project conforms to those standards. The initial standards of interest are:

- BPMN2

- ArchiMate
- TOGAF

4.3.3 Tooling

The tooling working group will be responsible for ensuring that the tools being developed as part of the project are supporting the methodology and compliance criteria as defined by the other working groups.

The requirements from a tooling perspective will derive from the initial framework description, outlined in this document, the other working groups, and the user community.

5 List of Contributors

We would like to thank the following contributors:

Gary Brown (gbrown@redhat.com)
Jeff DeLong (jdelong@redhat.com)
Jay Goode (jay.goode@cognizant.com)
John Graham (jgraham@redhat.com)
Glyn Humphreys (glyn.humphreys@cognizant.com)
Bhavish Kumar (bhavish.kumar@cognizant.com)
Mark Little (mlittle@redhat.com)
Sanda Morar (sanda.morar@cognizant.com)
Paul Mukherjee (paul.mukherjee@cognizant.com)
Andrew Porter (andrew.porter@cognizant.com)
Steve Ross-Talbot (steve.ross-talbot@cognizant.com)
Ricky Tapper (rickyscott.tapper@cognizant.com)