**Carnegie Mellon University**

MSE - 2009

**Software Architecture Document for “Global Software Development Tool”**

**Team “The Jazz Rockers”**

**Nan Li, Abhishek Minde, Jeff Salk, Yuki Saito, Bhanu Sistla**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Changed by | Modifications | Version |
| 03/15/09 | The Jazz Rockers | Skeleton of the SAD created. | 0.0 |
| 03/26/09 | The Jazz Rockers | Added project goals and architecture drivers | 0.1 |
| 03/31/09 | The Jazz Rockers | Added tactics and rationale | 0.15 |
| 04/01/09 | The Jazz Rockers | Added architecture views and prose | 0.2 |
| 04/03/09 | The Jazz Rockers | Revised the architecture | 0.3 |
| 04/11/09 | The Jazz Rockers | Revised the function requirement list | 0.4 |
| 04/25/09 | The Jazz Rockers | Revised the function requirement list | 0.5 |
| 05/02/09 | The Jazz Rockers | First version placed under configuration-control. | 1.0 |
| 05/02/09 | The Jazz Rockers | Renamed based on naming convention. | 1.1 |
| 06/02/09 | The Jazz Rockers | Removed redundant “GSD” from name. | 1.2 |
| 06/09/09 | The Jazz Rockers | Renamed “Import Controller” to “Import Manager” | 1.3 |
| 06/09/09 | The Jazz Rockers | Renamed to put “(SAD)” in title. | 1.4 |
| 06/10/09 | The Jazz Rockers | Removed "WebServiceCallReturnConn" description from Import Plug-in section since it's not used by that plug-in. | 1.5 |
| 06/12/09 | The Jazz Rockers | * Clarified that Data Reader reads different types of external data, not just ASCII files. * Updated the Config Manager (Import plug-in) diagram to reflect that it potentially reads from local configuration files. * Corrected description of Config Manager (Import plug-in) | 1.6 |

Because this document is for academic evaluation, it could potentially to create a bias for individuals. Therefore, in the “changed by” column instead of writing the name of the individual that changed the document, we decided to write the team name.

Table of Contents

[1. Purpose of the Document 5](#_Toc228872126)

[2. Problem Statement 6](#_Toc228872127)

[2.1 Project Overview 6](#_Toc228872128)

[2.2 Project Background 6](#_Toc228872129)

[2.3 Business Drivers 6](#_Toc228872130)

[2.4 Project Goals 7](#_Toc228872131)

[2.5 Context Diagram 7](#_Toc228872132)

[3. Architectural Drivers 10](#_Toc228872133)

[3.1 Business Constraints 10](#_Toc228872134)

[3.2 Technical Constraints 10](#_Toc228872135)

[3.3 High Level Functional Requirement 11](#_Toc228872136)

[3.4 Quality Attributes 13](#_Toc228872137)

[Utility Tree 13](#_Toc228872138)

[Six-part Scenario 14](#_Toc228872139)

[4. Architectural Style/Patterns & Tactics 16](#_Toc228872140)

[4.1 Styles/Patterns 16](#_Toc228872141)

[Predominant Style 16](#_Toc228872142)

[Styles/patterns used in refinements 17](#_Toc228872143)

[4.2 Tactics Used 18](#_Toc228872144)

[5. Architecture Overview 22](#_Toc228872145)

[5.1 Component and Connector View of the GSD System 22](#_Toc228872146)

[5.2 Component and Connector View of the GSD Client 24](#_Toc228872147)

[5.3 Module views of the GSD System 41](#_Toc228872148)

[5.4 Allocation view of GSD System 42](#_Toc228872149)

[5.5 Alternate solutions 43](#_Toc228872150)

[Alternate solution1 43](#_Toc228872151)

[Alternate solution2 44](#_Toc228872152)

[Component and Connector View of the GSD Client 44](#_Toc228872153)

[Module view of the GSD Client 47](#_Toc228872154)

[5.6 Architectural decisions and quality attribute trade-offs 49](#_Toc228872155)

[Architecture Decision #1 (AD1): 49](#_Toc228872156)

[Architecture Decision #2 (AD2): 49](#_Toc228872157)

[Architecture Decision# 3 (AD3): 50](#_Toc228872158)

[Architecture Decision# 4 (AD4): 50](#_Toc228872159)

[Architecture Decision# 5 (AD5): 51](#_Toc228872160)

[Architecture Decision# 6 (AD6): 51](#_Toc228872161)

[Architecture Decision# 7 (AD7): 51](#_Toc228872162)

[Summary of tradeoff analysis: 52](#_Toc228872163)

[6. Architectural Analysis 53](#_Toc228872164)

[Quality attribute scenario QA5 - Modifiability: 53](#_Toc228872165)

[Quality attribute scenario QA3 - Availability: 54](#_Toc228872166)

[7. Conclusion 56](#_Toc228872167)

[7.1 Lessons learned: 56](#_Toc228872168)

[Architecture helped as a communication tool (client and team): 56](#_Toc228872169)

[Ports helped us to achieve consistency between different views 56](#_Toc228872170)

[Evolutionary architecture approach 56](#_Toc228872171)

[ACDM reviews 57](#_Toc228872172)

[8. Appendix 58](#_Toc228872173)

# Purpose of the Document

This document covers following:

* Client’s vision, project goals, context, and constraints of the system being developed
* Architectural drivers (high level functional requirements, quality attributes, and constraints) and prioritized utility tree
* Architectural views
* Architectural decisions
* Tradeoff analysis

The specification is intended for the following audience:

* Stakeholders for the Global Software Development (GSD) project
  + Team Jazz Rockers
  + Customer representatives: Marcelo Cataldo and Charles Shelton
  + Developers at client’s company
* MSE Faculty
* Audience that might want to get an insight into or analyze the architecture and the design of the GSD tool.

# Problem Statement

## Project Overview

* Project name: Global Software Development Tool
* Clients: Marcelo Cataldo

Charles Shelton

* Mentors: Phil Bianco (SEI)  
   Gil Taran (CMU)
* Developer team: Nan Li  
   Abhishek Minde

Jeff Salk

Yuki Saito

Bhanu Sistla

## Project Background

Our client has several globally distributed business units. Different business units use different tools to collect and manage software related data. The client has identified the following key issues that have resulted from this current state of the business:

* Different business units have different processes for collecting software engineering data (metrics, artifacts etc.).
* Making meaningful observations about the collected data is hard.
* Collected data in one phase of a software engineering process is not always used in downstream processes.

The vision of the project is to develop abstracted representation of data coming from different tools and software processes. In addition, multi-modal views of real time data are provided to make project teams more aware of the project and the processes. The tool can also maintain relationships between multiple project data elements, illustrating how they are related to each other and their impact on each other. It provides basis to organize historical data for future developments.

## Business Drivers

In order to alleviate aforesaid issues our client has come has a vision to build a tool which would be able to

* Enhance developers' productivity by reducing communication overhead when obtaining accurate status on project artifacts in a geographically distributed software development environment
* Facilitate team collaboration by allowing accurate and immediate impact analysis upon changing one or more software engineering data and by avoiding expensive rework in a geographically distributed software development environment
* Enable stakeholders to monitor and track different tasks in projects as well as the overall accurate status of each project at any point in time

## Project Goals

* To abstract data elements that are managed by IBM Jazz Platform
* To establish traceability between different abstracted data elements
* To provide traceability views to different stakeholders
* To import fixed-format client company’s data into Jazz repository

**Note:** By data we mean the entire project related information that is being created, updated, retrieved or deleted in the Jazz repository during various phases of the project. The data represents documents, source files, status reports, defect logs, architecture and design documents, specifications, and all other kinds of artifacts. It also includes the information on resources, timeline, progress, schedules, budget, estimation, planning and reporting data.

## Context Diagram

Following figure depicts the context of the project.

**Figure 2.5.1 Context diagram of the GSD tool**

**Description of Context Diagram**

The above context diagram represents the interaction of the GSD tool with the existing Jazz Server and RTC components. The GSD tool is being built as an extension to the RTC Client which has Jazz technology (Server and Repository) as its underlying framework.

In the above context diagram, the component in blue called ‘GSD Tool’ is the one that we are developing. The components in brown represent the existing system. The user interacts with the GSD tool as well as the RTC client. Thus, the primary objective of the GSD tool is to extend the capabilities of the Jazz Server. The user shown in the context diagram can belong to one of the two categories:

***Process Person****:*

*Process Person* is the one who defines the abstraction of data elements stored in the Jazz repository which in turn is based on Jazz server. The data elements stored in the Jazz server are referred to ‘Jazz element’. An abstraction of the ‘Jazz element’ as defined by the process manager is known as an ‘abstract data element’.

These abstract data elements form the basis on which traceability is defined. Traceability can be defined as a relationship between two abstract data elements. A defined traceability can be uni-directional or bi-directional, meaning that if traceability has been defined from requirements to defects as a bi-directional relationship, then the users would be able to see all the requirements that are related to a particular defect and vice versa. Also, a traceability can also be defined between two traceabilities, thus enabling a process manager to be able to define a chain of relationships between related abstract data elements, using the principle of transitivity (if A is related to B, and B is related to C, then A is related to C).

***Normal Users:***

Other users who use the tool to create their role-based reports or extract a specific view from the tool (process person is a normal user with special authorization to be able to define define abstraction and traceability). This category of users makes use of the capabilities that are provided by the tool.

The objective behind defining traceabilities between different abstract data elements is to be able to generate different role based views. based on the role(s) assigned to each user, he/she will have access to view specific categories of traceability instances. Thus, different users with different roles are assigned to different views that show traceabilities between different elements.

Another important capability of the tool is to provide compatibility between some of the existing tools (like ClearQuest, Doors etc.) and the GSD tool. GSD tool can import data (exported from the existing tools) from files having a predefined format (Word, CSV, and Excel) and store it in the Jazz repository. Thus, the data stored in external tools would be exported to a specific file format that GSD tool understands, and then store it in the Jazz repository.

***Example****:*

A process manager wants to assign an integrator tester (a normal user) his role based view that shows the tests on which that specific user should work and the requirements covered by each test (like in traceability matrix). In order to do this, the process manager first defines an abstract data element called ‘requirements’ that would be based on the Jazz element of type ‘requirement’ in the Jazz server. Similarly, he creates another abstract data element called ‘test cases’ (of type ‘tests’ in Jazz element) and yet another one called ‘integrator tester’ (of type Resource in Jazz Element). He then defines a relationship between requirements to test cases and another relationship between test cases to integrator tester. The GSD tool would automatically define a transitive relationship between integrator tester and requirements. Now, the process manager defines a view containing the traceability that he/she defined in the previous step. This view in turn is assigned to the specific user of that particular role. Now when an integrator tester logs onto the system, the tool presents him a view that shows all the test cases he/she is suppose to be working on along with the requirements that each test case is related to. In order to present this view, the GSD tool accesses the abstraction of Jazz data which in turn is formed based on the abstract data definition that is stored in the Jazz repository. This interaction happens through the RTC client. In this way, our GSD tool provides extensibility to the existing Jazz server.

# Architectural Drivers

## Business Constraints

Following are business constraints of the project:

The project period is from August 25, 2008 to December 15, 2009. Due to the structure of the MSE program, the deliverables will be ready by August 1, 2009. After the delivery, the Jazz Rockers team will resolve defects in the deliverables (no additional functionalities or features) until December 15, 2009.

The Jazz Rockers team has no budget for commercial software. Any third-party software used must be free.

Resource availability is defined below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Academic Semester** | **Period** | **#weeks** | **#Engineers** | **#Hrs/Week**  **(Per Eng.)** | **Total Hours** |
| Fall 08 | August 25 – December 12 | 16 | 5 | 9 | 720 |
| Spring 09 | January 12 – May 10 | 17 | 5 | 12 | 1020 |
| Summer 09 | May 18 – August 9 | 12 | 5 | 48 | 2880 |
| Fall 09 | August 24 – December 14 | 18 | 5 | 12 | 1080 |
| Total Hours | | | | | 5700 |

## Technical Constraints

|  |  |
| --- | --- |
| **ID** | **Description** |
| TC1 | The tool shall use Jazz Platform for accessing data from Jazz Team Server. |
| TC2 | The tool must be delivered as a plug-in to IBM Rational Team Concert. |
| TC3 | The tool shall run on all the platforms on which IBM Rational Team Concert can run – specifically, Windows and Linux. |

## High Level Functional Requirement

The high level functional requirements of this project have been classified into four categories

1. Data abstraction
2. Traceability views
3. Importing data
4. Authorization
5. Data Abstraction

|  |  |
| --- | --- |
| **ID** | **Description** |
| FR1 | The system shall allow the user to define new abstract data element or create a new data element that represents an existing data element in Jazz data repository. |
| FR2 | The system shall allow an abstract data element to be associated with Jazz repository’s internal data elements (e.g. work item, defect, etc.). |
| FR3 | The system shall allow an abstract data element to be associated with a file revision in the Jazz repository. |
| FR4 | The system shall allow selecting certain attributes of Jazz data elements to be a part of abstract data element. |
| FR5 | The system shall allow the system's data module to read data items from the Jazz repository component. |
| FR6 | The system should allow a transformation filter to be associated with an abstract data element. |

1. Traceability Views

|  |  |
| --- | --- |
| **ID** | **Description** |
| FR7 | The system shall allow the user to create a relationship between two abstract data elements. The relation is called as a traceability definition. |
| FR8 | The system shall allow the user to create a new relationship of existing two traceability definitions (chaining of relationships). |
| FR9 | The system shall allow an authorized user to view instances of a selected traceability definition. Instances are elements that are participating in a selected traceability. Let’s call this resulting view as traceability view. |
| FR10 | The system shall be able to represent complex data elements that contain primitive and other complex data elements. Requirements, change requests, metrics will be modeled as complex abstract data elements. |
| FR11 | The system shall display the traceability view visually. |
| FR12 | The system shall represent the bidirectional traceability between requirements and components, components and work packages, requirements and work packages, work packages and test cases, requirements and related requirements, requirements and design documents, customer requests and requirements, the design and code/implementation, components and the responsible people who are working or had worked on the components. |
| FR13 | The system shall represent the traceability from requirements to the related discussion or clarifications. |
| FR14 | The system shall share traceability definitions among authorized users. |
| FR15 | The system shall allow user to copy and paste the data of the traceability view in the table. |
| FR16 | The system should allow the user to define different sorting criteria in a traceability view. |
| FR17 | The system should allow the user to define his filter criteria in the traceability view. |
| FR18 | The system should allow user to export the table of the data of traceability view to CVS file. |
| FR19 | The system should allow user to export the graphical view of traceability to image. |

1. Import Data

|  |  |
| --- | --- |
| **ID** | **Description** |
| FR20 | The system shall import a fixed format comma separated file into Jazz repository. The mapping of fields in a CSV file to the elements in the Jazz repository is known. |

1. Authorization

|  |  |
| --- | --- |
| **ID** | **Description** |
| FR21 | The system shall allow a process manager to assign roles to the defined users. |
| FR22 | The system shall allow authorization on the abstract data elements in the data module. |
| FR23 | The system shall share abstract data definitions only with authorized users |
| FR24 | The system shall allow a process manager to grant read/write privileges on traceability definitions to the users. |
| FR25 | The system may allow an authorized user to engage elements in a selected traceability definition. |

## Quality Attributes

A quality attribute workshop (QAW) was conducted to elicit the required qualities. We identified and prioritized 17 quality attribute scenarios. Based on the priorities we refined the top 7 scenarios, which we will consider during architecting the GSD tool. We will also keep the other scenarios in mind; however, we will evaluate our architecture based on these 7 refined scenarios.

### Utility Tree

The following quality attributes drive the design of architecture. Each quality attribute scenario is ranked with importance (I) defined by the clients, and level of difficulty (D) estimated by the team and the client. Both values are based on a scale of High (H) – Medium (M) – Low (L).

**Figure 3.4.1 Quality attribute utility tree**

### Six-part Scenario

Following are two of the defined QA scenarios. Due to space limitation, we did not include all scenarios in this section.

1. QA5: Modifiability

**Table 3.4.1 Six-part Scenario for QA5: Modifiability**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Attribute | | Scenario | | Importance | Difficulty |
| QA5 | Modifiability | | The BU has a new analysis to be visualized using the subset of data elements existing in the repository after the system is deployed. The new analysis feature should be implemented, tested, and ready for use within 5 person days. | | H | M |
| Scenario  Components | | Stimulus Source | | The BU has a new analysis to be visualized using the subset of data elements existing in the repository. | | |
| Stimulus | | The BU has a new analysis to be visualized using the subset of data elements existing in the repository. | | |
| Artifact | | The new analysis feature requested by the BU | | |
| Environment | | The system is deployed. | | |
| Response | | The new analysis feature should be implemented, tested, and ready for use within 5 person days. | | |
| Response  Measure | | A person day | | |

1. QA 3: Availability

**Table 3.4.2 Six-part Scenario for QA3: Availability**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Attribute | | Scenario | | Importance | Difficulty |
| QA3 | Availability | | The GSD tool experiences its internal software error and crashes. The system is operational. The state of the previously applied operation is restored on restart and the last saved data should be retained and the impact of crashes must be contained in the GSD tool. | | H | H |
| Scenario  Components | | Stimulus Source | | The GSD tool | | |
| Stimulus | | The GSD tool experiences its internal software error and crashes. | | |
| Artifact | | The GSD tool | | |
| Environment | | The system is operational. | | |
| Response | | The state of the previously applied operation is restored on restart and the last saved data should be retained and the impact of crashes must be contained in the GSD tool. | | |
| Response  Measure | | Restoration time, data loss, and the number of components outside the GSD tool that are affected by crashes | | |

# Architectural Style/Patterns & Tactics

This section discusses the architectural styles and patterns that the GSD tool’s architecture possesses. It also provides rationale for selection of these styles.

## Styles/Patterns

As the GSD system is bound to have more than one style, we first identified the predominant style for the tool. And then we have decomposed various components in the predominant style into further refinements. These refinements components are based on other architectural styles.

### Predominant Style

We analyzed our architecture drivers and we decided to use Client and Server style. Following is the rationale for choosing client and server as the predominant style for the GSD tool.

* The technical constraint (TC1), the Jazz platform, has a client and server architectural style. As we are extending the Jazz server, we realized that Client and server style is the perfect match for the GSD tool.
* Abstract data definitions and traceability definitions needs to be shared with all authorized users. Users can access these definitions from globally distributed locations and number of users may range from 50 to 1000. In order to achieve these requirements and qualities, we decided to have a GSD server that would store and manage abstract data and traceability definitions.
* The GSD client would be responsible for fetching these definitions from the GSD server. The GSD client is also responsible for fetching the required data from the Jazz server. This means that the GSD client plays client’s role for two different servers, GSD Server and Jazz Server.

**Figure 4.1.1 Client and server style for GSD tool**

In this solution, GSD server and Jazz server are two different servers. GSD server has the responsibility to store data abstraction definitions and traceability definitions. GSD client reads these definitions and uses RTC client to read data for these definitions from the Jazz server. E.g. traceability definition, requirement to component is stored in the GSD server; however, while populating the traceability view for a particular requirement, the data for this traceability is retrieved from the Jazz server. This architecture would allow centralization of definitions.

### Styles/patterns used in refinements

##### Event systems

**Implicit invocation for graphical**

Eclipse has the user interface frameworks GEF/GMF (Graphical Editing Modeling Framework), and the GEF framework uses implicit invocation style. If a view element changes it raises an event to notify other affected view elements and controllers. For example, Changes in the elements in GSD editor/view (traceability views) will invoke events and will be listened by corresponding event handlers. Thus, the flow of the program is determined by the events.

**Implicit invocation for invoking Rational Team Concert functions**

Some of the RTC functions are invoked using events. This would also allow the low coupling between GSD client and RTC client.

##### Call-Return

**Tiered Architecture:**

We decided to use Tiered architecture in bridging two elements, GSD client and RTC client. We will delve into details in later sections.

**Main program and subroutine**

In our plug-in, there are certain internal components that interact best through method invocation. For example, certain components will need access to the Jazz server data and GSD server data. This would be best handled using the given APIs.

**Object-oriented**

Eclipse plug-in architecture uses this style and since we are building eclipse plug-ins, it’s imperative to use this style while interacting with eclipse components. We will use GMF (Graphical Modeling Framework) for certain features that requires data models (such as forms), and we will use GEF for certain features like traceability views.

**Pipe-and-filter**

While performing the data abstraction, transformation of the concrete data (underlying Jazz data) would be required. And such chaining of transformation is possible through the use of defined filters. Pipe-and-Filter style would give the flexibility to arrange the transformation filter according to needs and it would also promote plugging in newly developed filters.

## Tactics Used

1. **Modifiability**

We considered the following modifiability tactics up to certain extent (team review satisfied).

**Localization of changes**

We grouped together the components that we anticipated to be affected by the similar kind of changes. E.g. we grouped the view entities together; data abstraction components together. We tried to generalize these modules based on their functions.

**Separation of concern**

Based on the functional requirements, we categorized elements into different sets of components. We also analyzed our technical constraint (TC1) and we decided to isolate GSD client from the details of the RTC client. This would provide better modifiability and also client may still use the tool with other platforms than Jazz.

**Defer binding**

We decided to use event oriented style for certain aspects of the GSD client. This would achieve the late biding, which would also allow plugging in the components without needing many changes in the GSD client.

1. **Availability**

As per the QA3, availability requires that if GSD server fails to operate, Jazz server should not be affected. Similarly, if the GSD client is malfunctioning, the Rational Team Concert client should not be affected. Instead, it should give an error message that GSD server/ GSD client is not behaving correctly.

We had discussed two architecture solutions for the GSD tool. In the first, we required the modification to the Jazz server. This modification would have involved server side components that would become a part of the Jazz Server. And the second solution was to create a separate GSD server, which would work independently from Jazz server. GSD client has the responsibility to interact with both the server.

We analyzed both the solutions; we discussed them with our clients. And we decided to go with the independent GSD server (solution2). Its implications are discussed in the architecture decisions section.

Following are the tactics that we considered in order to support the defined availability scenario in our architecture:

**Replication**

We have decided to run the GSD server in an application server. For availability of the GSD server, we are solely relying on configuration of the application server in which GSD server will run. Server replication and load balancing would be handled by the application server. The GSD server will be designed in such a way that it supports multiple server instances and is capable of swapping data between instances (J2EE features).

**Ping and echo**

Our components can have an inbuilt ping and echo (this won’t be event based, rather it would be a call-return communication) mechanism such that the components are mutually responsible for to detect the availability of other components. We are also considering establishing this mechanism to be organized in a hierarchy such that the each component determines (supervises) the availability of the components that works under and report any failure to the component that is supervising it. This is better than having a remote fault detector that would ping all the components which would have required coupling all components with the fault detector (affecting the modifiability).

**Exception mechanism**

We would have exception handlers that would be responsible for handling exceptions occurred in every component. Each component would have its own exception handler that would be inherited from a (Super Exception Handler class). Each handler would catch relevant exceptions and return corresponding messages back to the caller component. The messages would be framed in such a way as to ensure the clarity and understandability to a novice user. This would thus promote our QA 18: Usability.

**CheckPoint/RollBack**

We would use check point and roll back mechanism to recover from an erroneous state to a stable state. This would prevent the user from executing a long running task again when it fails. Check point is a recording of a consistent state, so when our GSD system would fail in an unusual manner with an undetected inconsistent state (especially when it’s fetching the data from the jazz server), the system would be restored using a previous check point of a consistent state and a log of transactions that occurred since the last check point. We would provide the Logging mechanism that would keep track of the Stack traces to all the operations. The Log file would help the user detect the source of error and also trace back the sequence of operations to the cause of error.

1. **Performance**

We have decided to use caching of data at different levels. At the client side (eclipse) we have discussed caching the definitions until they are updated in the server.

We are going to have data pools (Beans) at the server side using Enterprise Java Beans, to maximize the performance for fetching of abstract data element and traceability definitions.

We cannot improve the performance of the data retrieval from the Jazz server, as we are not able to perform any changes to it.

For QA17, we have thought about having separate plug-ins for different file-types and file formats. This plug-in mechanism would allow the developers to plug-in new import mechanism for data in new format. This would lower the reusability; however, the performance scenario (QA17) would be achieved.

1. **Testability**

We did not consider this QA15 while creating this architecture, because, in the QAW it was a low priority scenario, which we considered during the review with the client to be of higher priority. However, we’ve thought about having logging feature for every component.

1. **Usability**

**Long running operations**

As most of the data retrieval operations in the GSD tool are going to take time, we are going to design the UI components such that they do not halt when certain long running operations are in progress. We will design asynchronous invocations for such operations.

**Estimated time of operation**

Our architecture would support estimation time required for a requested user operation. GSD server would maintain a history of past performances, based on which user requests will be estimated. This means, the tool would provide the user feedback such as “the requested operation might take 2 minutes”.

**Maintain a model of the task**

The model of the task to be performed would be maintained to determine context, such that GSD system can have an idea of what the user is attempting and can provide assistance. For example, the users would not need to re-enter the same data again, once the data is obtained for the first time, the related information like project name, its components, resources etc, would be pre-populated. This would save time and resources at the users end to perform a task.

**Support user initiative**

We would design system responses (from UI perspective) and enumerate responsibilities of the system to respond to user commands. For example, if the user cancels a previous action, the GSD system would listen, and inform the components that are collaborating with the cancel command.

**Design time tactics**

Separating user from the rest of the application: we are using Model view controller (MVC) pattern. E.g. GSD UI views form the view, GSD action handlers form the controllers and Data abstraction manager, traceability manager and Import Manager form the model.

# Architecture Overview

Following sections describes different views of the GSD system architecture.

## Component and Connector View of the GSD System

**Figure 5.1.1 High level C&C view of the GSD System**

The figure 5.1.1 depicts the client and server style. GSD client is a client of GSD server and Jazz server. Components in green are the existing components which are used by the GSD system.

Following table describes responsibilities of the different elements.

**Table 5.1.1# Element Catalog of “GSD System”**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Jazz Server | This is the existing component. Jazz server maintains the repository, called as **“Jazz Repository**”, in which data such as files, work-items, team information etc is stored. RTC client access Jazz server. GSD tool needs to create an abstraction of this data. And then the GSD tool can link different data abstraction with the help of traceability. Links between different data elements are also stored on the Jazz server. These links are used in forming the traceability between the elements connected by the links. Jazz server also maintains user information including username and password. It provides authentication services. |
| RTC Client | RTC client runs in an eclipse platform. It accesses the data from Jazz server. It can also upload and update the data stored on the Jazz repository. It provides several API’s to access the data stored in the Jazz server. RTC client also performs authentication of the user. |
| GSD Server | Primary responsibility of the GSD server is to maintain abstract data type and traceability definitions. These two types are definitions form the basis of different traceability views. However, the actual data for which abstract data type is defined resides on the Jazz server. |
| GSD Client | GSD client is based on Eclipse platform. It is a collection of plug-ins. It needs assistance from the RTC client regarding the data stored on the Jazz. It performs the following functions.   * It accesses abstract data type and traceability definitions stored on the GSD server. And based on the definitions and user requests, it retrieves data from the Jazz server via RTC client. * GSD client can also import data from certain types of disk files into the Jazz repository. * It visualizes different traceability views requested by the user * It uses RTC client’s authentication services to determine the user and his roles. This user information is used while preparing role based traceability views. |
| Traceability config files | Traceability views can be configured using the traceability config files, which are stored in XML. |
| Traceability Files | Traceability view plug-in can export the traceability data that is displayed on the view to disk files, traceability files. User can opt to export the traceability view to be exported to a disk file. |
| Data files | Data files are the files that need to be imported in the Jazz Repository. Different Import plugins recognize different file format. E.g. CSV Import plugin can only import data stored in CSV files. |

**Table 5.1.1# Connector used in “GSD data plug-in”:**

|  |  |
| --- | --- |
| **Connector** | **Purpose** |
| GSD-RTC Bridge | This connector is the most important connector. It allows the GSD client to interact with the RTC. GSD client can access the data which is specific to Jazz using this connector. GSD Client does not know details of the Jazz platform and the bridge is solely responsible to map GSD elements to RTC elements and vice versa. This bridge transforms data coming from Jazz server in the format that is recognized by GSD and vice versa. |
| WebServiceCallReturnConn | This connector represents a web service call made by a caller to a callee. The information is transferred over http(s) connection. |
| FileReadWriteConnT | This connector allows a “user” role to read from or write to a disk file. |
| EventConnT | This represents an event bus. Event providers put events on the event bus which are received by the concerned event listeners. |

## Component and Connector View of the GSD Client

Following diagram depicts the high level architecture of the GSD client.

**Figure 5.2.1 High level C&C view**

We have three types of plug-ins which will be integrated in the eclipse platform where RTC client has been integrated too*. GSD client* and *GSD Server* are the GSD components that will be developed by us. RTC client and RTC server are the existing components (technical constraint) which we are not going to modify. GSD data plug-in is responsible for data abstraction and traceability management. Traceability view plug-ins are responsible for displaying different traceabilities. Import plug-ins do the job of importing the data from fixed format files into the Jazz repository. The following element catalog describes responsibilities of these components.

**Table 5.2.1# Element Catalog of “GSD data plug-in”**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Jazz Server | This is the existing component. RTC client access Jazz server. Jazz server maintains the repository in which data such as files, work-items, team information etc is stored. GSD tool needs to create an abstraction of this data. And then the GSD tool can link different data abstraction with the help of traceability. Links between different data elements are also stored on the Jazz server. These links are used in forming the traceability between the elements connected by the links. |
| RTC Client | RTC client runs in an eclipse platform. It accesses the data from Jazz repository. It can also upload and update the data stored in the Jazz repository. It provides different API’s to access the Jazz server data. |
| GSD Server | Primary responsibility of the GSD server is to maintain abstract data type and traceability definitions. These two types are definitions form the basis of different traceability views. However, the actual data for which abstract data type is defined resides on the Jazz server. |
| GSD data plug-in | This is an eclipse plug-in. It is responsible to create and manager definitions of abstract data elements and traceabilities. This plug-in interacts with the Rational Team Concert via GSD-RTC bridge. And it accesses definitions stored on the GSD server over webservices (WP1 port). GSD data plug-in can upload new definitions of abstract data types and traceabilities on the GSD server.  It first fetches the definitions from the GSD server, and then it fetches the required data (as defined in the definition) from the Jazz server via RTC client. |
| Traceability View Plug-ins | We can have more than one traceability plug-ins. Each traceability plug-in is responsible to visualize a particular traceability (e.g. “requirement to work packages”, “components to test cases”, etc. ). Traceability plug-ins can also export the traceability data into disk files. Traceability plug-in fetches traceability related data from the GSD data plug-in through the port P1. |
| Import Plug-ins | Import plug-in is responsible for reading a disk file for a particular format and upload the read data in the Jazz repository. One of the Import plug-ins the CSV Importer, which is responsible to read CSV records from a disk file and map the fields to their corresponding data elements in the Jazz repository. This mapping is limited to certain data elements in Jazz and can be done using the UI views of the plugin.  ***Import plugins do not deal with concerns related to data semantics.*** It user of the import plug-in makes sure if the mapping and file formats are correct. |
| Traceability config files | Traceability views can be configured using the traceability config files, which are stored in XML. |
| Traceability Files | Traceability view plug-in can export the traceability data that is displayed on the view to disk files, traceability files. User can opt to export the traceability view to be exported to a disk file. |
| Data files | Data files are the files that need to be imported in the Jazz Repository. Different Import plugins recognize different file format. E.g. CSV Import plugin can only import data stored in CSV files. |

The following tables describes the connectors used in the diagram 5.2.1

**Connectors:**

**Table 5.2.1# Connector used in “GSD data plug-in”**

|  |  |
| --- | --- |
| **Connector** | **Purpose** |
| GSD-RTC Bridge | This connector is the most important connector. It allows the GSD client to interact with the RTC. GSD client can access the data which is specific to Jazz using this connector. GSD Client does not know details of the Jazz platform and the bridge is solely responsible to map GSD elements to RTC elements and vice versa. This bridge transforms data coming from Jazz server in the format that is recognized by GSD and vice versa. |
| WebServiceCallReturnConn | This connector represents a web service call made by a caller to a callee. The information is transferred over http(s) connection. |
| FileReadWriteConnT | This connector allows a “user” role to read from or write to a disk file. |
| EventConnT | This represents an event bus. Event providers put events on the event bus which are received by the concerned event listeners. |
| CallReturnConnT | Caller calls a certain method of callee and callee performs the requested oprtation and returns the result back to the caller. |

Now, we will look at the refined C&C view of the GSD Data Plug-in.

**Figure 5.2.2 Refined C&C view of GSD data plug-in**

The fig 5.2.2 depicts the internals of the GSD data plug-ins. It consists of some UI views using which user interact with the system. User can create and modify definitions of abstract data types and traceabilities. GSD Data plug-in is also responsible to fetch definitions from the Jazz server and actual data from the Jazz server. The following element catalog describes responsibilities of different sub-components of the plug-in.

**Table 5.2.2# Element Catalog of “GSD data plug-in”**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| View(s) | View is responsible for displaying screens in the GSD client. These screens primarily includes, abstract data definition and traceability definition screens and popup menus. These UI views communicate with their corresponding handlers via eclipse runtime (menu selection bus). These notifications are handled by the handlers. |
| Handler(s) | Handlers receive events generated by the UI Views. Each event will have its own handler. Action handlers invoke methods of GSD Façade based on the user actions. They also pass the data present in the event to the GSD Façade. |
| GSD UI Delegate | GSD UI delegate is responsible for invoking the screens of GSD client. Traceability and Definition manager can invoke the screens by the help of GSD UI. If a request for RTC UI comes from a certain component it uses RTC GSD Bridge to bring the RTC screen up. |
| GSD Facade | GSD Façade hides the complexities of certain components (data abstraction manager, traceability manager and Import Manager) from the action handlers. Action handlers have access to only those methods that they need. This helps in lowering the coupling. |
| Data abstraction manager | * This component is the heart of the GSD client. It is responsible for reading and storing the abstract data definition types (ADT) on the GSD server. And then based on the definitions it fetches the corresponding data from the Jazz server. The data fetched by this component is used by traceability manager while displaying the traceabilities. * This component also creates new abstract data definitions. It uses certain views to present certain details and then it collects details from the user thought action handlers. This abstract data definition type (ADT) is stored in the GSD server. * This element is responsible to interact with the GSD server. |
| Traceability manager | * Traceability manager is responsible to define new traceabilities in the GSD server. It uses a collection of views and handlers to gather traceability definition details from the user and then updates the definition into the GSD server. * Traceability manager is also responsible for fetching the traceability definition from the GSD server. * It finds which abstract data elements are participating in a given traceability definition, and then it retrieves instances for those data elements from the jazz server. It uses Data abstraction manager to retrieve abstracted data elements from Jazz. |

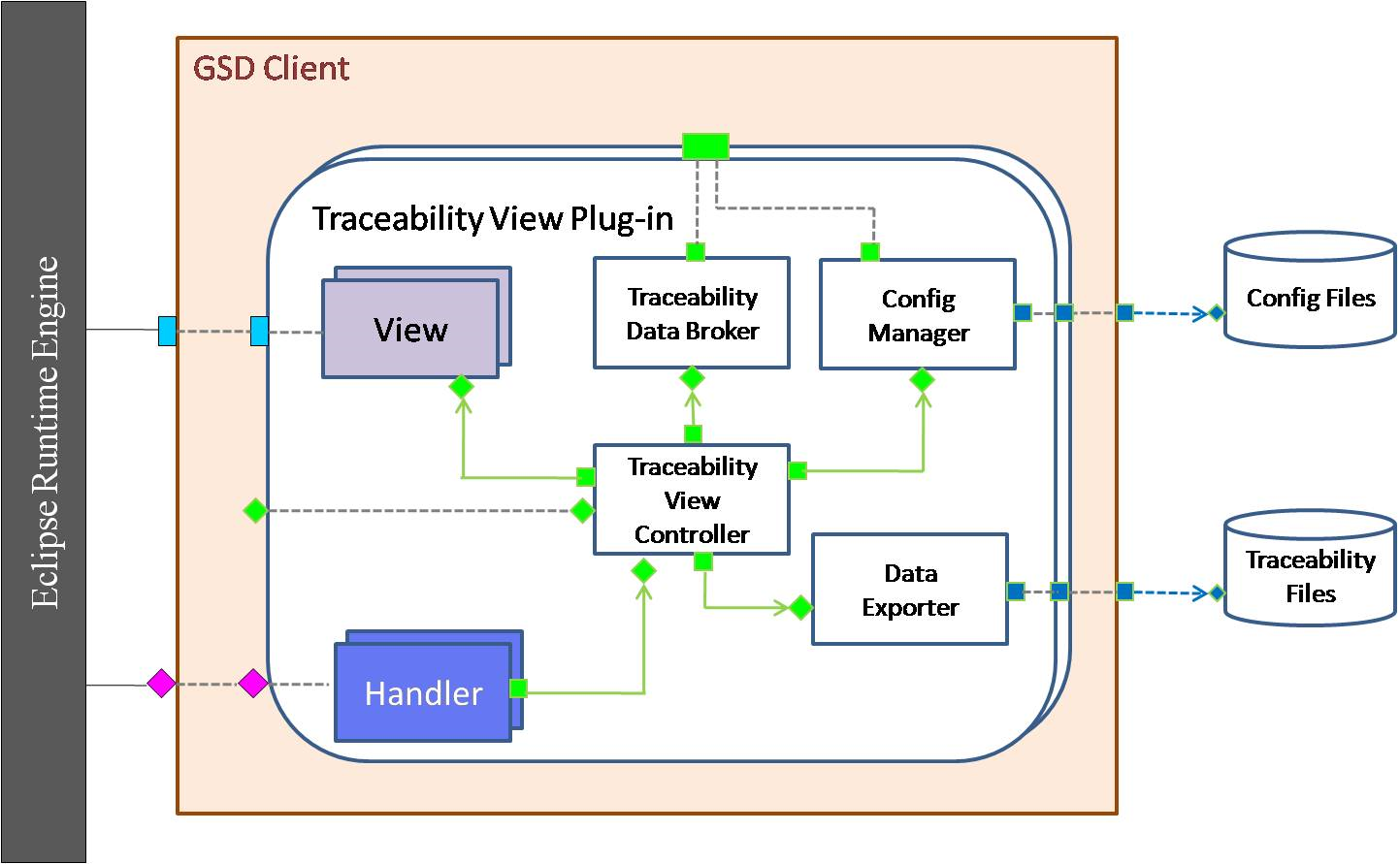
Now we will look at the GSD-RTC bridge, which is responsible to bind both GSD and RTC together. This is one of the most important connector.

**Figure 5.2.3 Refined C&C view of the GSD-RTC Bridge**

As shown in the figure 5.2.3, the bridge connector has a tiered architecture. This connector is responsible to bridge the gap between RTC client and the GSD client. Each tier has a certain responsibility which works towards to adaptation of data between two clients.

**Table 5.2.3 # Element Catalog of “GSD-RTC Bridge”**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Jazz specific Tier | All components in this tier directly talks to the RTC client through exposed APIs. E.g. Team broker is responsible for getting the team area information. RTC client provides API’s for such functionalities. |
| Data interpreter Tier | This tier is responsible to transform GSD specific data queries into Jazz specific queries and vice versa. This GSD client sends queries in terms of XML. |
| GSD Director Tier | This tier is responsible for handling all the GSD client’s requests for data. |
| Logger | Logger logs all performance analysis and exceptions. Logger can be turned On or off by the administrator. |
| Performance Analyzer | Performance analyzer measures latency time for all Jazz API’s used in all the components in the Jazz specific tier. It also measures how much time it takes to transform GSD queries into RTC queries. This performance data is written to a log file, which can be later used.  Performance analyzer can be turned On or Off based on the need. |
| Bridge administrator | Bridge administrator controls and supervises all the components in all the tiers. It can turn on and off the logging and performance measurement features. |

****

**Figure 5.2.4 Refined C&C view of Import plug-in**

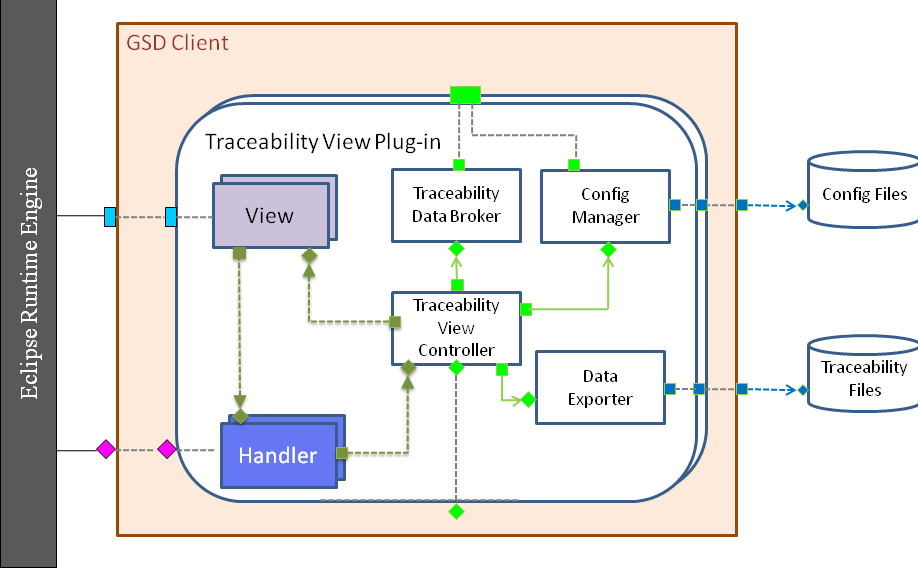
As shown in the figure 5.2.4, Import Manager reads from the data files. Import Manager parses the files based on the known format and puts the data on the eclipse runtime engine. This data is picked up by the GSD plug-in and GSD plug-in has the responsibility to upload the data in the Jazz repository.

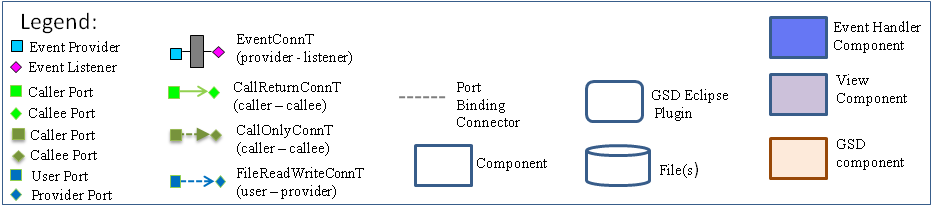
**Table 5.2.4 Element Catalog of “GSD Import Plug-in”**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| View(s) | View is responsible for displaying screens in the GSD client. These screens primarily include import screen and popup menus. These UI views communicate with their corresponding handlers via eclipse runtime (menu selection bus). These notifications are handled by the handlers. |
| Handler(s) | Handlers receive events generated by the UI Views. Each event will have its own handler. Action handlers invoke methods of Import Manager based on the user actions. |
| Import Data Broker | Import Data Broker communicates with GSD Data-plug-in to upload the data into the Jazz repository. |
| Import Manager | Import Manager works as a hub, communicating with other components, View(s), Handler(s), Import Data Broker, Config Manger, and Data Reader, to achieve successful parsing the imported files based on the known format and upload the parsed data into Jazz repository. |
| Config Manager | Config Manager is responsible for uploading/reading the mapping definition files to/from (respectively) the GSD Server (calling on the Data plug-in) or by using data files stored locally on the same machine. |
| Data Reader | Data Reader reads external data, such as ASCII files (e.g., comma-separated value (CSV) files) or potentially even database records. Different types of files and data formats need different Data Reader import plug-ins. |

**Table 5.2.4 Connector used in “GSD Import Plug-in”**

|  |  |
| --- | --- |
| **Connector** | **Purpose** |
| FileReadWriteConnT | This connector allows a “user” role to read from or write to a disk file. |
| EventConnT | This represents an event bus. Event providers put events on the event bus which are received by the concerned event listeners. |
| CallReturnConnT | Caller calls a certain method of callee, and callee performs the requested operation and returns the result back to the caller. |





**Figure 5.2.5 Refined C&C view of traceability plug-in**

Traceability views are initialized with data fetched by the traceability view controller from the GSD data plugin. This data is either shared or transferred from the GSD data store. Traceability view controller is responsible to handle and its representation based on the selected filters. Filters are selected by user on the screen. There might be some RTC screens that are invoked from the GSD traceability screens. View controller forwards the request for the RTC screen through RTC GSD Bridge. This isolates the RTC details from the core logic.

**Table 5.2.5# Element Catalog of “Traceability View Plug-in”**

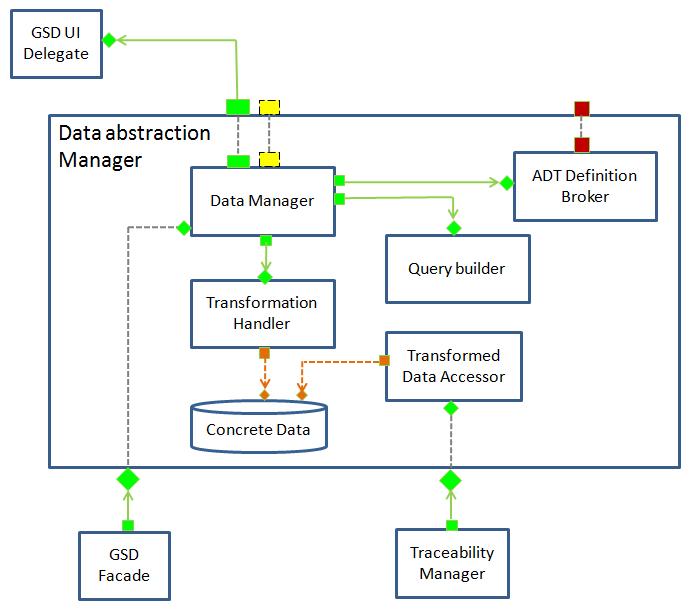
|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| View(s) | View is responsible for displaying screens in the GSD client. These screens primarily include traceability graphics screens showing different relationship between elements and popup menus. These UI views communicate with their corresponding handlers via eclipse runtime (menu selection bus). These notifications are handled by the handlers. |
| Handler(s) | Handlers receive events generated by the UI Views. Each event will have its own handler. Action handlers invoke methods of Traceability View Controller based on the user actions. |
| Traceability Data Broker | Traceability Data Broker communicates with GSD Data-plug-in to get the data required to generate the traceability views the user requests. |
| Traceability View Controller | Traceability View Controller works as a hub, communicating with other components, View(s), Handler(s), Traceability Data Broker, Config Manger, and Data Exporter, to achieve successful representations of traceability views to the user. |
| Config Manager | Config Manager is responsible for handling the configuration for traceability views according to the user’s request, such as sorting view, filter view. |
| Data Exporter | Data Exporter exports traceability view to the corresponding type of file according to the user’s request. The types of the exported file include CSV file or image. |

Following are the connectors that are used in the 5.2.5 figure.

**Table 5.2.5# Connector used in “GSD Traceability View Plug-in”**

|  |  |
| --- | --- |
| **Connector** | **Purpose** |
| WebServiceCallReturnConn | This connector represents a web service call made by a caller to a callee. The information is transferred over http(s) connection. |
| FileReadWriteConnT | This connector allows a “user” role to read from or write to a disk file. |
| EventConnT | This represents an event bus. Event providers put events on the event bus which are received by the concerned event listeners. |
| CallReturnConnT | Caller calls a certain method of callee, and callee performs the requested operation and returns the result back to the caller. |
| CallOnlyConnT | Caller calls a certain method of callee, and callee performs the requested operation but does not return a result (directly) to the caller. This is a subset of the CallReturnConnT connector type, encompassing only the “call” part of a “call-return” connection. |

Figure 5.2.6 depicts the refined C&C view of the data abstraction manager. This view helps to understand how the abstract data is processed.



**Figure 5.2.6 Refined C&C view of data abstraction manager (DAM)**

As shown in the figure, DAM is responsible for storing data in Jazz and retrieving data from Jazz server. It also fetches ADT definition from the GSD server.

**Table 5.2.6# Element Catalog of DAM**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Data element Definition Manager | Data element definition manager is responsible for following things,   * Retrieve definitions of abstract data element from the GSD server. * Update, create abstract data element on the GSD server. * It also configures transformation handler based on the transformation filters defined in the data definition. * Based on the abstract data elements, with the help of query builder it prepares a query to retrieve the required attributes. * It sends the data queries to the RTC GSD Bridge, which transforms these queries into Jazz specific queries. Then transformed query fetches data from the Jazz server. This data is return back the Definition manager. |
| Transformation handler | Transformation handler is configured by the definition manager. It transforms the data coming from the definition manager in the format specified by the definition. This data is stored in the data pool. |
| Data pool | Data pool stores transformed data for a given abstract data element. Traceability manger uses this data to prepare traceability views. |
| Query Builder | Based on the properties and attributes defined in the abstract data definition, query builder prepares a generic query in XML that is understandable by the RTC GSD bridge. This query is independent of Jazz details. |

Now, we will see where authorization on different ADT definition is achieved. The figure 5.2.7 depicts the refinement of C&C view of ADT definition broker.

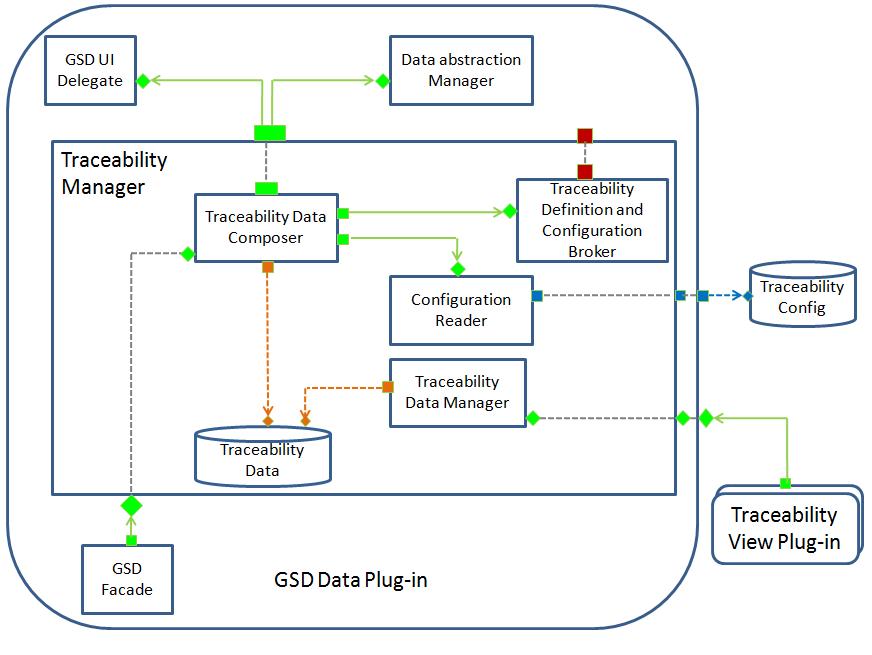
**Figure 5.2.7 Refined C&C view of ADT definition broker**

Following is the element catalog of the ADT definition broker.

**Table 5.2.7 Element catalog of ADT definition broker**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Abstract Data Type Definition Broker | Abstraction data type definition broker fetches and provides abstract data definitions either from the GSD server or from the definition cache. It also maintains data consistency between the GSD server and the definition cache. |
| Definition  Cache | Definition cache improves performance of providing the abstract data definition. Whenever a new abstract data type definition data is fetched from the GSD server, it is stored in this definition cache so that the next time the same abstract data type definition is requested, it is provided by the definition cache rather than it is fetched from the GSD server. |
| Authorization Verifier | Authorization verifier verifies whether or not the user with a certain role is authorized to access the requested abstract data type definition. |

Following figure depicts the refined C&C view of the Traceability Manager.



**Figure 5.2.8 Refined C&C view of traceability manager**

The following table describes the responsibilities of the different elements in the figure shown above.

**Table 5.2.8 Element catalog of traceability manager**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Traceability Data Composer | Traceability data composer reads and integrates traceability definition data and its configuration, both of which are linked to the abstract data element by the data abstraction manager. |
| Traceability Definition and Configuration Broker | Traceability definition and configuration broker downloads from and uploads to the GSD server the traceability definition and traceability configuration. |
| Configuration Reader | Configuration reader reads traceability configuration such as sorting criteria and filtering criteria and returns it to the traceability data composer. |
| Traceability Data Manager | Traceability data manager provides the traceability view plug-in with the traceability data, which is fetched from the traceability data repository. |
| Traceability Data | Traceability data stores actual traceability data in it, whose format is XML. |

**Figure 5.2.9 Refined C&C view of Traceability and Configuration definition broker**

**Table 5.2.9 Element catalog of Traceability and Configuration definition broker**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Traceability Definition Broker | Traceability definition broker fetches and provides traceability definitions either from the GSD server or from the definition cache. It also maintains traceability data consistency between the GSD server and the definition cache. |
| Traceability Configuration Broker | Traceability configuration broker uploads from and downloads to the GSD server traceability configuration data. Traceability configuration data includes sorting criteria or filtering criteria, both of which becomes useful when the user wants to see the traceability view according to their visual preference. |
| Definition Cache | Definition cache improves performance of providing the traceability data definition. Whenever a new traceability definition data is fetched from the GSD server, it is stored in this definition cache so that the next time the same traceability definition data is requested, it is provided by the definition cache rather than it is fetched from the GSD server. |
| Authorization Verifier | Authorization verifier verifies whether or not the user with a certain role is authorized to view the requested traceability view. |

**Component and Connector View of the GSD Server**

**Figure 5.2.10 Refined C&C view of GSD Server**

As shown in the fig 5.2.10, the GSD server will store primarily three types of information, configuration, performance data and definitions. These definitions, configuration and performance data is shared across all GSD clients. Following table describes the responsibilities of different components.

**Table 5.2.10 Element catalog of GSD Server**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Performance Estimator | Performance estimator records the performance data such as response time for different operations in the performance database. The historic data is used for estimating similar types of (similar load) requests. |
| Traceability Definition manager | Traceability definition manager maintain the database of traceability definitions. |
| Abstract data Definition manager | Abstract definition definition manager maintain the database of abstract data definitions. ADT represent the abstraction of the data. |
| Shared Configuration manager | Shared config manager stores simple configuration of traceability views such as “sort by” defined by the users. This configuration can be shared across different GSD users. |
| GSD Server Facade | This component shields the complexity of the GSD server and presents a simple interface to the external world. GSD clients interact with the GSD Server over web services. |

## Module views of the GSD System

Following figure depicts the module view of the GSD Data plugin.

**Figure: 5.3.1 Module view of GSD Client**

This is a module view (static) for the GSD Data Plugin. Colored modules are existing modules, which will be used by the GSD modules. GSD RTC Bridge module uses RTC modules. And “GSD UI Views and Dialogs” uses eclipse GEF/GMF. GSD action handler module uses the handler infrastructure of eclipse. Plugin configuration module will have plugin configuration XML.

We have a similar looking module diagrams for other plugins. Due to the space constraint we did not include them in the report.

## Allocation view of GSD System

**Figure 5.3.1 Allocation view of GSD System**

This allocation view shows how the GSD system will be deployed and what are the relationship between different elements and the environment they are running in. As shown in the figure 5.3.1, we will have four elements, Jazz Server Instance, GSD Server Instance, RTC Client, and GSD Client, within which RTC Client and GSD Client will be run on the same PC in Eclipse environment while Jazz Server and GSD Server might be deployed in different web/application servers (they could be deployed as two different applications on the same application/web server as well). We plan to use WebSphere Application Server (WAS) for both Jazz and GSD server. RTC Client or GSD Client will communicate with the corresponding servers over secured web services. There could be more than one computer that is running “RTC and GSD clients”.

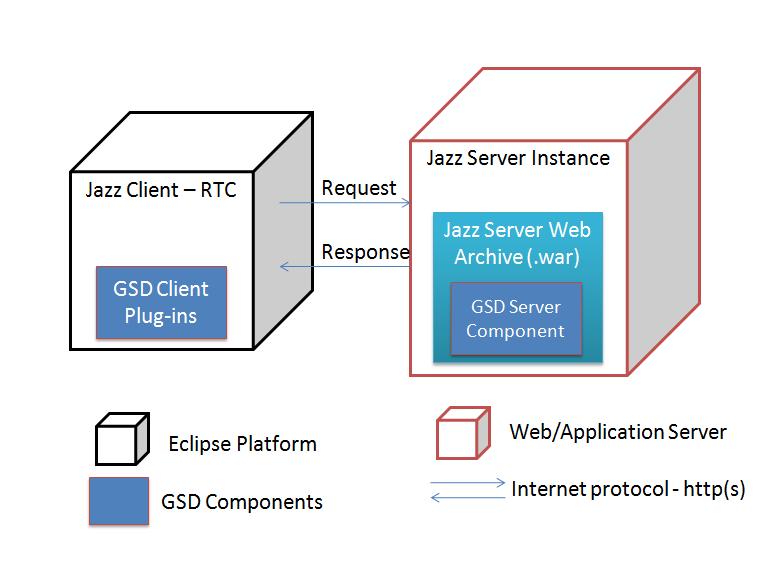
## Alternate solutions

### Alternate solution1

We have an alternate solution for the GSD tool. Although, this alterative is at a very higher level, we reckoned it to be very critical while making the decision. We have also alternative designs at refined levels. However, the following alternative would change the entire structure of the system.

**Figure 5.5.1 C&C view of the alternate solution (GSD server embedded in the Jazz Server)**

As showing in the fig 5.4.1, GSD server will be an extension to the Jazz server. GSD client plug-in would not directly call the GSD server; the data will be traveled through the intermediate tiers (Jazz RTC client and Jazz server). However, this would require a change to the jazz server and in future whenever any change is done to the GSD server; it would be required to bring down the Jazz server, which is almost infeasible in the production environment, as thousands of software developers would be connected using it. Following deployment view depicts the allocation of different GSD tool components on the machines.

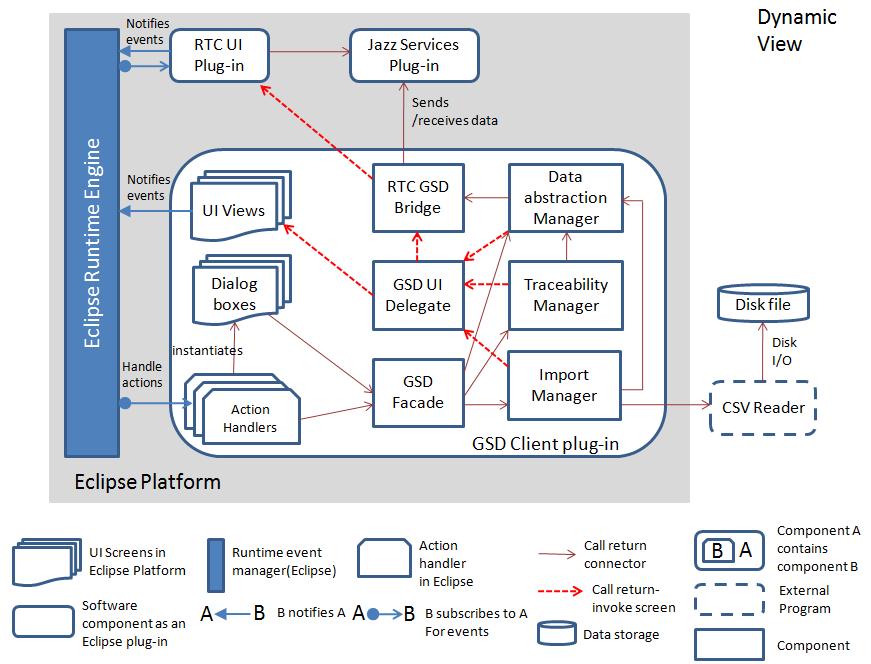


**Fig 5.5.2: Deployment view of the alternate solution #1**

### Alternate solution2

In this solution, the client side architecture had only one plug-in. However, due to the highest priority quality attribute, Extensibility, we did not go with this architecture.

### Component and Connector View of the GSD Client

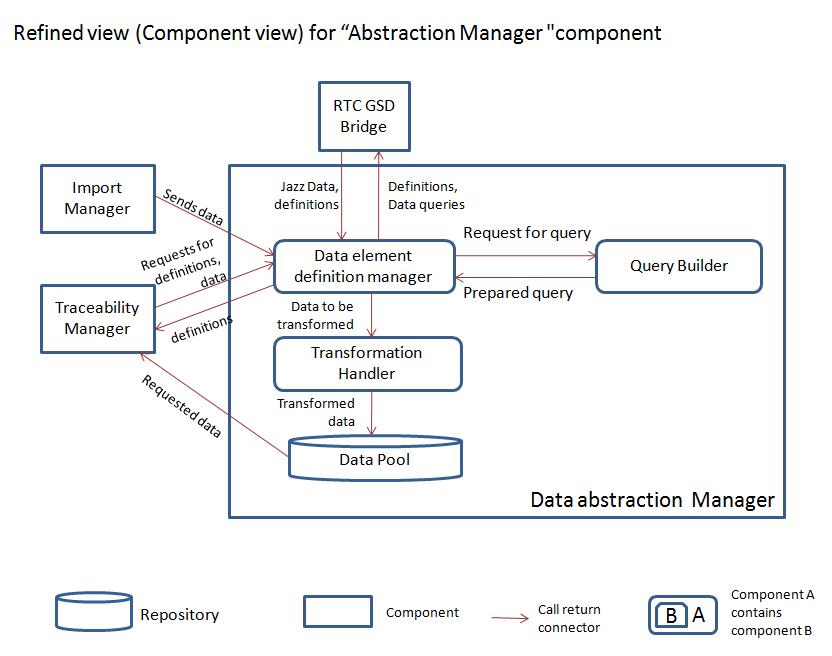


**Figure 5.3.3 C&C view of the GSD client (High Level)**

This is a component and connector view for the GSD client. GSD Client plug-in is an eclipse plug-in which will interact with eclipse framework and RTC plug-ins. Following table describes the components and their responsibilities.

**Table 5.3.3# Element Catalog of the GSD client (High Level)**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| UI View | UI view is responsible for displaying screens in the GSD client. These screens primarily includes, abstract data definition screens, traceability screens, traceability views. These UI views will post notifications in the eclipse runtime engine. These notifications will be handled by the action handlers. |
| Dialog Box | Dialog boxes show the notifications and some dialogues will also take simple input from the users. |
| Action Handler | Action handler receives events generated by the UI Views. Each event will have its own action handler. Action handlers invoke methods of GSD Façade based on the user actions. They also pass the data present in the event to the GSD Façade. |
| RTC GSD Bridge | This is one of the most critical components. This is responsible for isolating RTC details from the other GSD components. It’s the mediator between the RTC and GSD. It is responsible for invoking the RTC views. It also sends and receives data from the RTC Jazz plugins. It deals with the real Jazz data and it transforms it into the representation that is known to the Data abstraction manager. |
| GSD UI Delegate | GSD UI delegate is responsible for invoking the screens of GSD client. Traceability and Definition manager can invoke the screens by the help of GSD UI. If a request for RTC UI comes from a certain component it uses RTC GSD Bridge to bring the RTC screen up. |
| GSD Facade | GSD Façade hides the complexities of certain components (data abstraction manager, traceability manager and Import Manager) from the action handlers. Action handlers have access to only those methods that they need. This helps in lowering the coupling. |
| Data abstraction manager | * This component is the heart of the GSD client. It is responsible for reading the abstract data definitions from the GSD server. And then based on the definitions it fetches the data from the Jazz server. The data fetched by this component is used by traceability manager while displaying the traceabilities. * This component also creates new abstract data definitions. It uses certain views to present certain details and then it collects details from the user thought action handlers. This abstract data definition is stored in the GSD server. * This element is responsible to interact with the GSD server. |
| Traceability manager | * Traceability manager is responsible to define new traceabilities in the GSD server. It uses a collection of views and handlers to gather traceability definition details from the user and then updates the definition into the GSD server. * Traceability manager is also responsible for fetching the traceability definition from the GSD server. * It finds which abstract data elements are participating in a given traceability definition, and then it retrieves instances for those data elements from the jazz server. It uses Data abstraction manager to retrieve abstracted data elements from Jazz. |
| Import Manager | Primary function of the Import Manager is to import data from the CSV file. The file format is fixed and Import Manager knows which fields from the file maps to which field in the Jazz data element. It creates Jazz data elements based on the file data. |



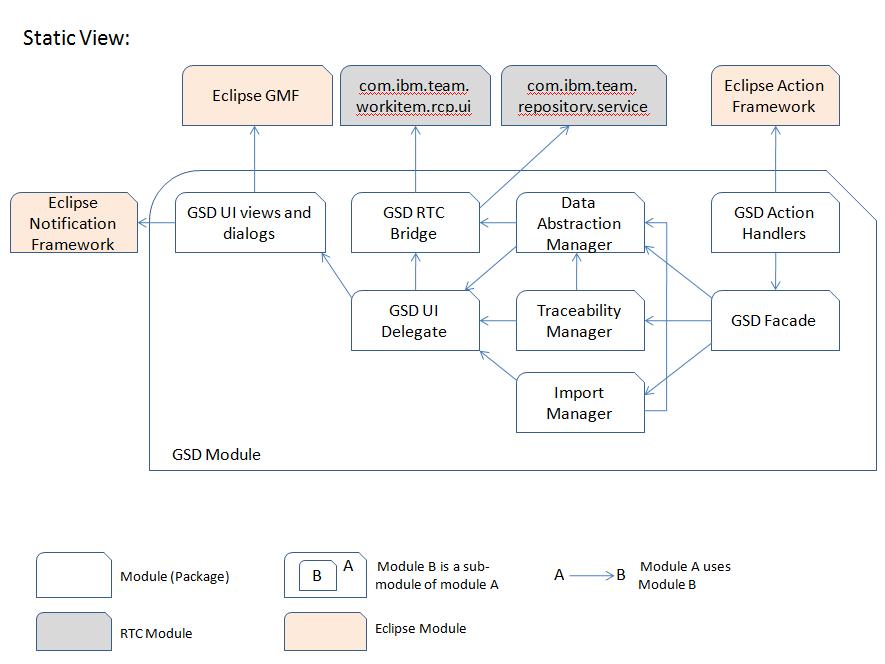
**Figure 5.3.4 C&C view of abstraction manager**

This is a refined component and connector view for the “Abstraction Manager” component shown in the figure 5.3.4

**Table 5.3.4 Element catalog of abstraction manager**

|  |  |
| --- | --- |
| **Component** | **Responsibility** |
| Data element Definition Manager | Data element definition manager is responsible for following things:   * Retrieve definitions of abstract data element from the GSD server. * Update, create abstract data element on the GSD server. * It also configures transformation handler based on the transformation filters defined in the data definition. * Based on the abstract data elements, with the help of query builder it prepares a query to retrieve the required attributes. * It sends the data queries to the RTC GSD Bridge, which transforms these queries into Jazz specific queries. Then transformed query fetches data from the Jazz server. This data is return back the Definition manager. |
| Transformation handler | Transformation handler is configured by the definition manager. It transforms the data coming from the definition manager in the format specified by the definition. This data is stored in the data pool. |
| Data pool | Data pool stores transformed data for a given abstract data element. Traceability manger uses this data to prepare traceability views. |
| Query Builder | Based on the properties and attributes defined in the abstract data definition, query builder prepares a generic query that is understandable by the RTC GSD bridge. |

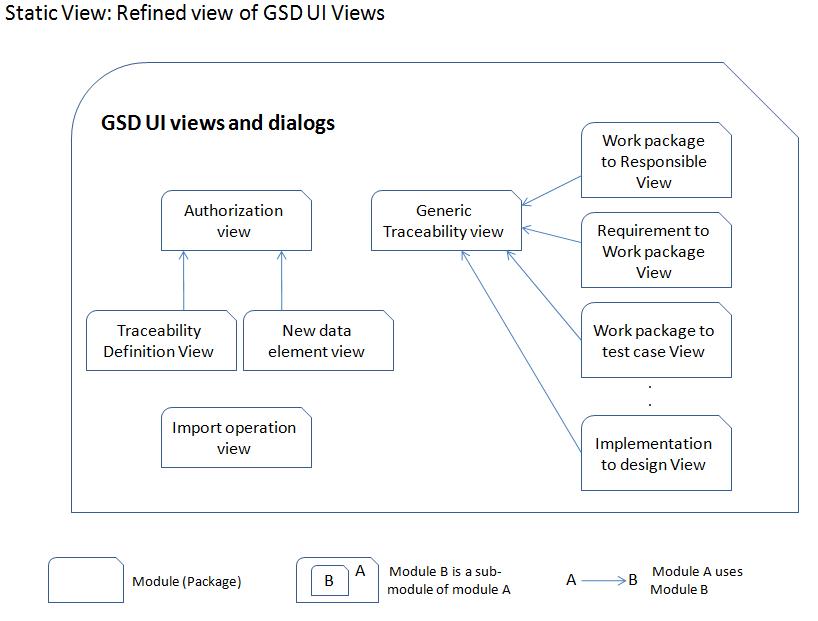
### Module view of the GSD Client



**Figure 5.3.5 High level module view of the GSD client**

This is a module view (static) for the GSD client. GSD Module depicts the modules to be developed for GSD client. Colored modules are existing modules, which will be used by the GSD modules. GSD RTC Bridge module uses RTC modules. And “GSD UI Views and Dialogs” uses eclipse GEF/GMF. GSD action handler module uses the handler infrastructure of eclipse.

Following figure is the refined view of the “GSD UI Views and dialogs” package.



**Figure 5.3.6 Refined module view of “GSD UI Views and Dialogs”**

As shown in the figure, we will have a “Generic Traceability View” that could be extended by specific traceability views. “Authorization view” module is responsible for representation of authorization view of the users before they can access certain traceability views. “New data element” package contains views for creating a new data element. Traceability Definition view module is responsible for the UI for creating a new traceability definition.

## Architectural decisions and quality attribute trade-offs

The GSD architecture evolved through four alternative architectures. We made several architectural decisions. The following are the key decisions.

### Architecture Decision #1 (AD1):

We made an architectural decision to have a GSD server which will store the definitions. The alternative was to store the definitions locally on the GSD client as configuration files. We decided to go with the server because; the definitions can be created and shared between several GSD users working globally. This will improve the maintainability of the abstract data type and traceability definitions. If a process manager changes a definition, it would reflect on all the GSD clients, and there is no necessity to distribute the configuration files for traceability as it was with the alternative approach.

**Tradeoffs**:

1) **Modifiability**: This promoted the modifiability and upgradability scenarios (QA#17). With this approach GSD can be upgraded to new definitions at runtime.

2) **Performance**: This inhibits the performance scenario (QA#8). The client will first have to initially download the definitions from the GSD server, which will add to the latency time, affecting the performance.

### Architecture Decision #2 (AD2):

Initially, we thought to integrate the GSD server as a part of the Jazz server. The Fig 5.5.1 depicts the C&C view and the Fig 5.5.2 depicts the deployment view of the alternative architecture. In this architecture we thought that, both the definitions (abstract data type and traceability) and data would be stored in the Jazz server. However, we have a quality attribute scenario **QA#18** related to “Availability” quality. According to this approach, due to high coupling, any change in the GSD server might have required some changes in the Jazz server. And also, problems or crashes in the GSD server might have caused problem in the Jazz server, which is not at all acceptable as it might bring down the Jazz server, to which thousands of users might be connected.

Hence, we decided to separate the GSD server and store only the definitions on the GSD server. The data is still stored in the Jazz server. And GSD client is responsible to utilize definitions and data from these two servers.

**Tradeoffs**:

1) **Availability**: This promoted the availability scenario (QA#3), as GSD server is completely separated from the Jazz server and will not cause any problem to the Jazz server.

2) **Performance**: This inhibits the performance scenario (QA#8). The client will first have to download the definitions from the GSD server and then it will need to download the corresponding data from the Jazz server. This would require more web services calls and more co-ordination. Hence, it would cost us additional **latency time**.

### Architecture Decision# 3 (AD3):

The GSD client needs interact with the Jazz sever and it retrieves data from the Jazz server. We decided to hide jazz specific details from the GSD client (information hiding) and we came up with a GSD-RTC Bridge connector. This connector has a responsibility to transform GSD data/queries into Jazz specific data/queries and vice versa. GSD Client does not know anything about the Jazz details.

This provides flexibility, that in future, if the client wants to port the GSD System to another Jazz like system, they would need to replace this GSD-RTC Bridge with a new bridge connector, without having to change internals of GSD client/server.

And the other reason was, we did not have enough resources to learn about Jazz technology, and we wanted to **mitigate** **this** **risk**. Limiting the Jazz details on specific to this connector would shield the other parts from the risk that may arise because of incomplete knowledge about Jazz.

**Tradeoffs**:

1) **Modifiability**: This promotes the modifiability. Any change related to Jazz technology would not be rippled across all the architecture, and it will be limited only to the bridge.

2) **Performance**: This inhibits the performance scenario (QA#8). The GSD request for Jazz data needs to go through several tiers, because of the bridge connector. Hence, it would affect the latency time.

### Architecture Decision# 4 (AD4):

In one of the alternative architecture for GSD client we had all functionality of client in a single eclipse plug-in. However, we realized that this was no good for modifiability and upgradability scenarios. Any addition of new visualization of traceability would have required the change to the entire GSD client plug-in. Similarly, any new capability to import a new file format would have required the changes to GSD client plug-in. So, we decided to have separate plug-ins for import capability and traceability views (visualization). This architecture is flexible to accommodate new visualization and import capability in the form of new plug-ins. This would not required structural changes to the existing components.

**Tradeoffs**:

1) **Modifiability and upgradability**: This promotes the modifiability (QA#5) and upgradability (QA#17) scenarios. Any change related to Jazz technology would not be rippled across all the architecture, and it will be limited only to the bridge.

2) **Performance**: This inhibits the performance scenario (QA#8). Interaction and co-ordination between different plug-in requires an extra latency time; hence, this decision would inhibit the performance in terms of latency time required by end to end operations.

### Architecture Decision# 5 (AD5):

We decided to cache definitions in the GSD Data plug-in. “Traceability definition and configuration broker” (**Figure 5.2.8)** and ADT definition broker (**Figure 5.2.6)** cache the definitions until they are changed in the GSD server. This approach saved a web service call to the GSD server, as the definitions will only be retrieved from the server once. And they are used locally until the corresponding broker invalidates the local definition when they are changed in the GSD server.

**Tradeoffs**:

1) **Performance**: This promotes the performance scenario (QA#8). Latency time required by the web service call to fetch definition from the server is saved, as the definitions are initially downloaded locally.

### Architecture Decision# 6 (AD6):

We decided to observe the performance of different Jazz api’s in the GSD-RTC bridge. This performance data is logged and can be referred by the person who wants to analyze the performance of the bridge. Logging of exceptions is done in all major components.

**Tradeoffs**:

1) **Testability**: This promotes the testability scenario (**QA#18**). Difference performance and exception logs at bridge and other components would enhance the analysis of the problems, in turn enhancing the testability.

**2) Performance:** This inhibits the performance scenario (QA#8). Logging and performance measurement mechanism would cause an extra latency time, which in turn would inhibit the response time for different operations.

### Architecture Decision# 7 (AD7):

The GSD server keeps track of history of time required for different operations. It records the database of performance (latency time) for different sizes of operations such as retrieval of 100 work-items or 1000 work items. This data is used for estimating the future user requests and GSD client uses this data and displays the “Estimated time required for the operation” on the screen.

**Tradeoffs**:

1) **Usability**: This promotes the usability scenario (**QA#11**). Displaying the estimated time enhances the usability and user can cancel the operation if the time is too long or they can put the task in background.

**2) Performance:** This inhibits the usability scenario (QA#18). Performance measurement and estimation mechanism would cause an extra latency time, which in turn would inhibit the response time for different operations.

### Summary of tradeoff analysis:

Here’s the summary of architectural decision discussed in the previous section. We provide the following tables illustrating tradeoffs between important quality attributes. In each table, a ‘+’ indicates the attribute is promoted, a ‘-’ indicates the attribute is inhibited, and “blank” indicates the attribute is not affected by that architectural decision.

Table #: Trade-off Analysis on Architectures between Figure # and Figure #

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Quality Attribute | AD1 | AD2 | AD3 | AD4 | AD5 | AD6 | AD7 |
| QA5: Modifiability | + |  | + | + |  |  |  |
| QA3:  Availability |  | + |  |  |  |  |  |
| QA18:  Testability |  |  |  |  |  | + |  |
| QA8:  Performance | - | - | - | - | + | - | - |
| QA9:  Performance |  |  |  |  | + |  |  |
| QA17:  Upgradibility | + |  |  | + |  |  |  |
| QA11:  usability |  |  |  |  |  |  | + |

# Architectural Analysis

We performed four ACDM reviews on the evolving GSD architecture. By evolving we mean, we had considered several alternatives and based on the issues we identified in ACDM reviews, we performed some experimentation and corrected the architecture issues.

Our team walked through our architecture with our clients and checked traceability of all the functional requirements and quality attributes. Following are some of the key analyses. Due to limitation of space, we have not included all the analyses we performed on the architecture.

### Quality attribute scenario QA5 - Modifiability:

**Concern**: Addition of new traceability visualization

**Description**: The business unit (BU) has a new traceability to be visualized using the subset of data elements existing in the repository after the system is deployed. This new traceability should be implemented, tested, and ready for use within 5 person days

**Analysis**:

Add New Plugin

As shown in the above diagram, a new traceability plug-in can be added and GSD data plugin could be configured dynamically to use the new traceability view for a particular traceability definition.

### Quality attribute scenario QA3 - Availability:

**Concern**: **Availability when an internal error occurs**

**Description**: The GSD tool experiences its internal software error and crashes. The system is operational. The state of the previously applied operation is restored on GSD tool’s restart and the last saved data should be retained. The impact of crashes must be contained in the GSD tool and should not affect Rational Team Concert and Jazz server.

**Analysis**:

Fails

Still operational

Shields exceptions

As shown in the above diagram, if the GSD server fails, Jazz server is not affected. This is primarily because there is no direct dependency between jazz server and the GSD server. And also if the GSD client fails, the GSD-RTC Bridge shields propagation of the exceptions or errors to the RTC client, so that it still remains operational.

# Conclusion

## Lessons learned:

### Architecture helped as a communication tool (client and team):

We underwent four to five iteration cycles of architecture both within the team and with the clients. With each iteration, we identified new loopholes and gaps and gained a better understanding of the requirements of our tool. We have realized that architecture has worked so far to be the best communication tool for us to convey our understanding of the system to the clients and ensure that we both understand the system in the same way. Even within the team, having undergone the whole process of coming up with different architectural views has helped all the team members share their own perspectives and alternatives of how the system could be modeled, the tradeoffs related to each model that we create, the limitations that it may impose on the system and if the high priority quality attributes are being met. Designing the system architecture, has worked as a very effective requirement elicitation technique for our team, based on which we are now creating a refined SRS. Our reflection on the whole process tells us that architecture serves as one the very effective and efficient medium of communication between the clients and team members.

### Ports helped us to achieve consistency between different views

We did not think about port initially while designing the architecture. But after having a pre-review meeting with Dr. Garlan, we realized the importance of using ports and thinking about the architecture in terms of ports. Making use of ports to illustrate how different components would interact with each other through connectors helped us understand the practical aspects of the system clearly. For instance, we realized that we needed different kinds of ports to be able to communicate through different connectors, because of the difference in the type of communication. This not only helped us to think about how the user would interact with the system, but also as to how the data would flow from one component to another, thus helping us maintain a consistency between different views by knowing what kind of data would each component send and receive using which ports.

### Evolutionary architecture approach

As already mentioned above, our architecture evolved through multiple iterations of going through stages 4 and 5 of ACDM. The evolutionary approach proved to be very productive for the team since it resulted in three possible alternative solutions and 5 levels of drilling down into details of our final solution. We gained a balance between abstraction and refinement, in the sense; we realized the importance of keeping the architecture neither too abstract nor too refined. Our first level of architecture presents a very high level view and with each subsequent level it becomes more refined. Another important lesson to be learnt in the evolutionary approach was to know our exit criteria. Initially, we did have trouble finalizing our exit criteria, as to how to know when to stop, but as we went through review sessions, and could produce an end to end user scenario using the architecture, we found that our requirements getting stabilized and also it gave us an assurance of being on the right track since we could run through all our use cases and functional requirements using our architectural design.

### ACDM reviews

We followed the ACDM Review approach in four iterations, where in we went through the SAD (System architecture document), and QA scenarios and mapped those to the requirements. We performed our reviews along with our clients who were able to point out a few very important missing elements and inconsistencies. We discussed potential issues and concerns and noted those down in Issue Deposition Document. The solvable issues got resolved after the meeting but the irresolvable issues formed the base for experimentation. We conducted around 10 experiments based on the issues recorded during ACDM reviews. These experiments have helped us fill the technical gaps and design gaps. Our last ACDM review resulted in a ‘Go’ decision and we now are ready to proceed to the high level design phase.

# Appendix

**Quality Attribute Scenarios**

|  |  |  |
| --- | --- | --- |
| Scenario# | Quality Attribute | Description |
| #1 | Availability  [Use case] | The Global Software Development (GSD) tool cannot recognize the format of the external data source. The system is in its normal operational mode. Error message and abort. The GSD tool has no noticeable failures to the user. The Jazz client should still be operational. The user is notified of the error that is understandable. [tell them what type of error, have error codes, with statements] |
| #3 | Availability  [Use Case] | The GSD tool experiences its internal software error and crashes. The system is operational. The state of the previously applied operation is restored on GSD tool’s restart and the last saved data should be retained. The impact of crashes must be contained in the GSD tool and should not affect Rational Team Concert and Jazz server. |
| #5 | Modifiability  [Use Case] | The business unit (BU) has a new traceability to be visualized using the subset of data elements existing in the repository after the system is deployed. This new traceability should be implemented, tested, and ready for use within 5 person days. |
| #6 | Modifiability  [Growth] | The BU has a need to modify a subset of implemented features of the GSD tool. The system is deployed. The features will be modified and tested within ‘5’ person days per feature. |
| #7 | Modifiability  [Use Case] | The Jazz Repository component has been updated and the new component consists of features of interest to BU. The system is deployed. GSD tool should be able to get integrated with the new repository component within ‘5’ persons days. |
| #8 | Performance  [Use Case] | The user initiates an operation that involves visualization and traceability analysis of data from the repository. The system is operational. The GSD tool should complete the operation and the operation would display the result in 2 minutes amount of time for less than 30 traceability records. |
| #9 | Performance  [Use Case] | The user requests for import of a CSV file into the Jazz repository using the GSD tool. The system is operational as indicated in scenario #1 and the CSV file is available. The GSD tool completes the import operation and notifies the user of the success of operation within 5 minutes of time for the file having 10 columns and 500 rows. |
| #10 | Performance  [Use Case] | The user requests for importation of database records. The system is operational and the records are available in the external database. The GSD completes the operations and returns the records to the user within ‘5 sec’ amount of time for ‘20’ number of records. |
| #11 | Usability  [Use Case] | The user initiates an import or view traceability operation that takes an more than 1 minute of time. The system is operational as indicated in scenario #1. The GSD tool provides progress feedback to the user and gives provision of cancellation of operation to the user and also allows the user to make it a back ground process. |
| #12 | Usability  [Growth] | New employees hired at the BUs are required to use the GSD tool. The system is operational as indicated in scenario #1. The employees are familiar with eclipse. The users can learn how to use 75% of the features of the tool in ‘7’ number of days. |
| #13 | Security  [Use Case] | A user tries to access or modify data that he is not authorized to. The system Is operational as indicated in scenario #1. The GSD tool detects the non authorized access, logs it, denies the access of the data to the user and notifies the system administrator and the user with standard error messages. |
| #14 | Security  [Use Case] | A non authorized user tries to access connector specific credentials. The system is operational as indicated in scenario #1. The GSD tool detects the non authorized access, logs it, denies the access of the data to the user and notifies the system administrator and the user with standard error messages. |
| #15 | Testability  [Use Case] | A developer finds a bug and wants to debug the system through logs after the system is deployed. The tool allows the developer to trace the logs of inputs, stacks, configurations, past operations and data. The logs are presented in a readable and meaningful format. |
| #16 | Testability  [Use Case] | The user wants to perform a unit test on a component in the system without having to integrate to the rest of the system. The component to be tested exists and is already compiled. The user is able to create and run ‘10’ number of tests in ‘5’ number of person hours. |
| #17 | Upgradability  [Use Case] | A new release of the tool is available. The system is deployed. The upgrade is deployed without any loss of existing data and clearly defined process exists for performing the upgrade. |
| #18 | Testability  [User Case] | The user fails to perform an operation due to an error. The system is in operational state before execution of the operation. The system now is in erroneous state. The user should be able to get clear error messages indicating specific details as to what went wrong. The user should also be able to trace back the error to its origin by using stack traces and log data. |