

Project #5

SOFTWARE COLLABORATION FEDERATION

Version 1.0

Instructor: Dr. Jim Fawcett

Course: CSE681

Submitted by: Ojas Juneja

SU ID: 248597850

Contents

1. Software Collaboration Federation.....	4
1.1 Introduction	4
1.2. Overall Architecture Diagram	4
1.3 Users and Uses of SCF	6
2. Storage Management Subsystem	8
2.1 Architecture	8
2.2 Responsibilities of SMS	10
2.3 Storage of Data Structures for Servers using SMS	11
2.4 Critical Issues.....	12
3. Collaboration Server	13
3.1 Summary	13
3.2 Organizing Principles.....	13
3.3 Users and Uses	13
3.4 Responsibilities	14
3.5 Data Structure, Data Contract, Commands and XML Format for Collaboration Server	14
3.6 Architecture	16
3.7 Role of Agents in Collaboration Server	19
3.8 Critical Issues.....	20
3.8 Virtual Display System.....	20
3.8.1 Virtual Display Functionality	20
3.8.2 Architecture	22
4. Repository Server.....	22
4.1. Repository Server Overview	22
4.2 Organizing Principle	23
4.3 Users and Uses	23
4.4 Data Structure, Data Contract, Commands and XML Format.....	23
4.5 Architecture	26
4.6 Role of Agents in Repository Server.....	30
4.7 Critical Issues.....	30
5. Test Harness Server.....	31
5.1 Introduction	31
5.2 Organizing Principles.....	31

5.3 User and Uses	31
5.4 Data Structure, Data Contract, Commands and XML Format.....	32
5.5 Architecture	34
5.6 Role of Agents in Test Harness Server	37
5.7 Critical Issues.....	37
6. Build Server	38
6.2 Organizing Principles.....	38
6.3 User and Uses	39
6.4 Data Structure, Data Contract, Commands and XML Format.....	39
6.5 Architecture	41
7. Client	44
7.1. Client Overview	44
7.2 Roles and Responsibilities.....	44
7.3 Data Structures and XML Format.....	45
7.3 Architecture	46
7.4 Views	51
8. Policies	56
8.1 Asynchronous Behavior for Networking.....	56
8.2 Code Commit Policy.....	56
8.3 Handling Concurrent File Accesses	56
8.5 File Caching Policy.....	57
8.6 Check In policy	57
8.7 Notifications.....	57
8.8 Ownership.....	57
9. Services	58
10. Tools.....	58
10.1 Version Control Tools for Software Collaboration Federation	58
10.2 Testing Tools	59
10.2.1 Workflow.....	59
10.3 Bug Analysis Tool	60
11. Conclusion.....	61
12. References	61

1. Software Collaboration Federation

1.1 Introduction

Software Collaboration Federation(SCF) is a robust tool of combining individually tested software modules into the already existing system. In the software industry, there are multiple programmers working independently to build large software system. In such scenario, each developer work on different modules and at the end they integrate it to the existing system to make system productive. For handling this type of task and making software development a smooth process Software Collaboration Federation(SCF) is designed. SCF allows multiple clients to work in a collaboration and exchange project related information among each other. It is primarily concerned to ensure that new source code or modifications to an existing source code are successfully tested and checked-in and it is made extractable to all Client after they are successfully checked-in the immutable code repository. SCF is a client server architecture that consist of four servers and an arbitrary number of clients. Collaboration server that help to assign tasks to different team members and schedule meetings with the help of virtual display system, Repository Server stores all of the developing baseline, Test Harness Server runs sequence of tests on module that need to be check-in, Build Server provides build images of the source code that needed by the Test Harness Server for testing it before check-in actually take place. Client has user friendly GUI to access the functionalities of SCF. The most obvious users of this tool are Project Manager, Team Leader, Developer, Tester and Software Architect. They use SCF tool in different way such as for Check-in any module or Check-out any module, to test entire baseline, to view dependency between modules, to view reports of development activities, etc. At the last, the various critical issues of the SCF are discussed. They are,

1. Inefficient Workflow Management
2. Meeting Overlap
3. Concurrent File Commit
4. Unauthorized Repository Check-In
5. Server down while check-In
6. Heavy Load on Server to run Test Suites
7. Single Test taking lot of time to run

1.2. Overall Architecture Diagram

The following figure describes the architectural overview of Software Collaboration Federation (SCF). This architecture is consist of three servers and arbitrary number of clients. Overall idea behind each of this subsystems are explained below.

The main subsystems of SCF are as follows:

Client

Repository Server

Build Server

Test Harness Server

Collaboration Server

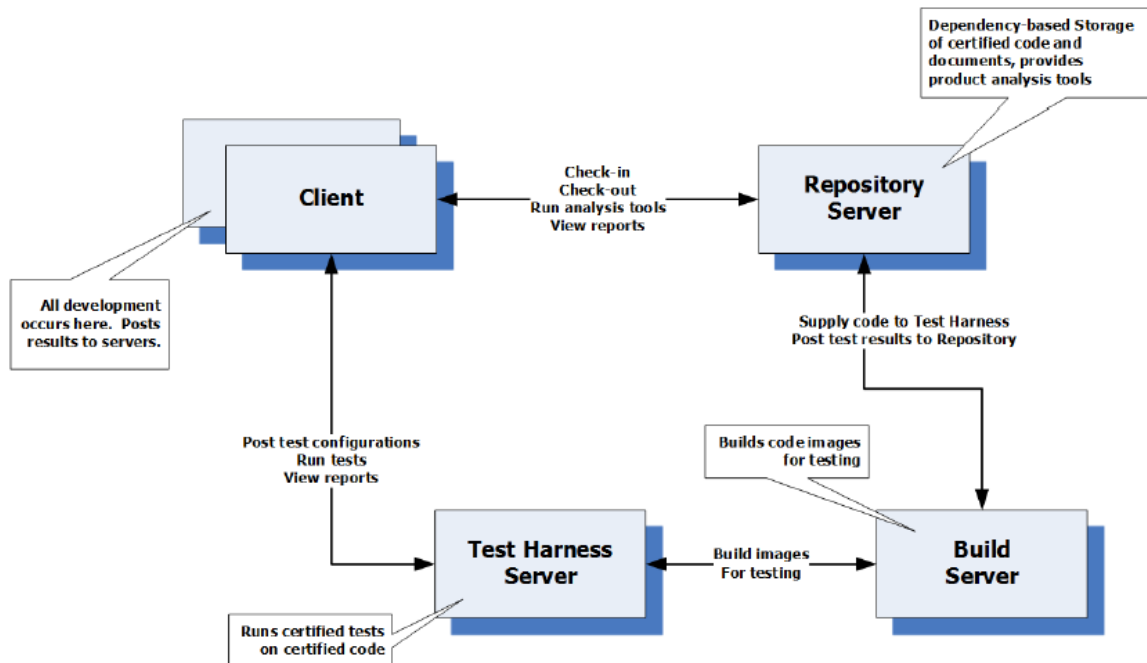


Figure 1 : Architectural Overview Of Contiguous Build And Integration System (CBIS)

[Reference: <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/lectures/Project5-F2015.htm>]

Client

Client is the initiation point of the subsystem. There can be multiple clients like software developer, tester, software architect, manager. All Clients have different privileges in our software system. The client has its own storage management subsystem which consists of event and data manager and no SQL database.

Main functions of the Client includes:

1. Client can check out the module from the Repository.
2. Client can check-in module to the Repository Server.
3. Client receives the result log from the Test-Harness Server & Repository Server.
4. Client can add/ retrieve work packages and provide tasks for different clients.
5. Client schedule meeting and retrieve meeting schedules.
6. Client can start running test cases by providing just configuration number which maps the test cases to that configuration number.

Repository server

Repository Server stores all developing baseline. Every updated module can be checked in into the repository by the authenticated user. Checking-in and Versioning of the modules are the most important tasks of Repository Server which we will discuss later. Main functions of Repository Server includes:

1. Stores modules as a collection of packages, generally it refers as baseline.
2. Supports check-in, check-out, extraction, and versioning of all modules.
3. It provides result to the Client regarding new module has been added to the baseline or not.
4. Generates XML file that represent the dependent packages of the requested module.

Build server

Build server compiles any code received from the Repository Server, which is required by the Test Harness Server to run the series of tests. Main functions of Build Server includes:

1. Accept build request message from the Test Harness Server.
2. Request the Repository Server for the needed source code for each build.
3. Store the build in the server.
4. Builds images as requested and give it back to the Test Harness Server if build is successful.
5. Maps old build with the build number so the client can run the old build.

Test Harness Server

It is responsible for configure and executing test suits provided by the Client. It generates the summary as a result log and sends it to the Client and Repository Server for notifying the client about the check-in status. The client is having a test management tool in which all the test cases written by the client is mapped to the test configuration number. The server side is also having a test management tool which has the same copy of clients. The client QC always synchronized with server QC. QC is the quality center (Test management tool). Main functions of Test Harness Server includes:

1. Frequent testing of new and changed repository source code by executing each test suites on its own thread.
2. Load test suits by providing test config number provided by the Client as an XML message.
3. Request to Build Server for the compilation of source code that resides into the Repository Server. They should be loaded based on the dependency order maintained by Repository Server.
4. Sends test results, test logs to Repository Server and Client.

Collaboration Server

Collaboration server provides important functionalities to the clients. It is involved in assigning tasks to clients using work packages and create meeting schedules. The meeting happens in virtual display system which also resides inside the collaboration server.

Main tasks of Collaboration server include:

1. Assigning work packages to team members and also checking their dependency.
2. Adding and retrieving meeting schedules.
3. Holding meeting on virtual display system.

1.3 Users and Uses of SCF

This section identifies the potential users of the Software Collaboration Federation (SCF). There are various users that can interact with the SCF in different ways. SCF is projected to be a perfect tool for following users.

Software Developers

Team Leader

Project Managers

Software Architects

Test Teams

Software developer

In any large software development project, there's a strong requirement to store the data. SQL db are not capable of storing of big data. So, many industries are moving towards. Software developers uses our No SQL database to do check-in and check-out as our NO SQL DB act as a repository. Uses the SCF to check their code by testing them and running on the server. They can get their meeting schedules.

Team leader

While dealing with the large software development project, there are number of teams, working toward the achievement of the objective of the software. Team Leaders have been assigned the group of developer to carry out scheduled task. Team Leader can check the performance of each developer of his team by monitoring the success ratio of adding/modifying the new modules by the developer. Team leaders can assign different packages to team members.

Project manager

All the team leaders work in concurrently and report their progress to the Project Manager. Project Manager is concern about the smooth functioning of the working software in order to meet specified requirement. Through SCF, Project Manager monitors the total number of successful tests that taken place and can also monitors the proper versioning of the different modules of the project, can also see which work packages are allocated to which person and can also see the completion of project by seeing the test cases run report.

Software Architect

When there's a requirement of adding unstructured data to the existing project, then SCF allows functionality to the Software Architecture to store data in the No SQL database which can store unstructured data in key value pairs thereby solving the problems of big data.

Test team

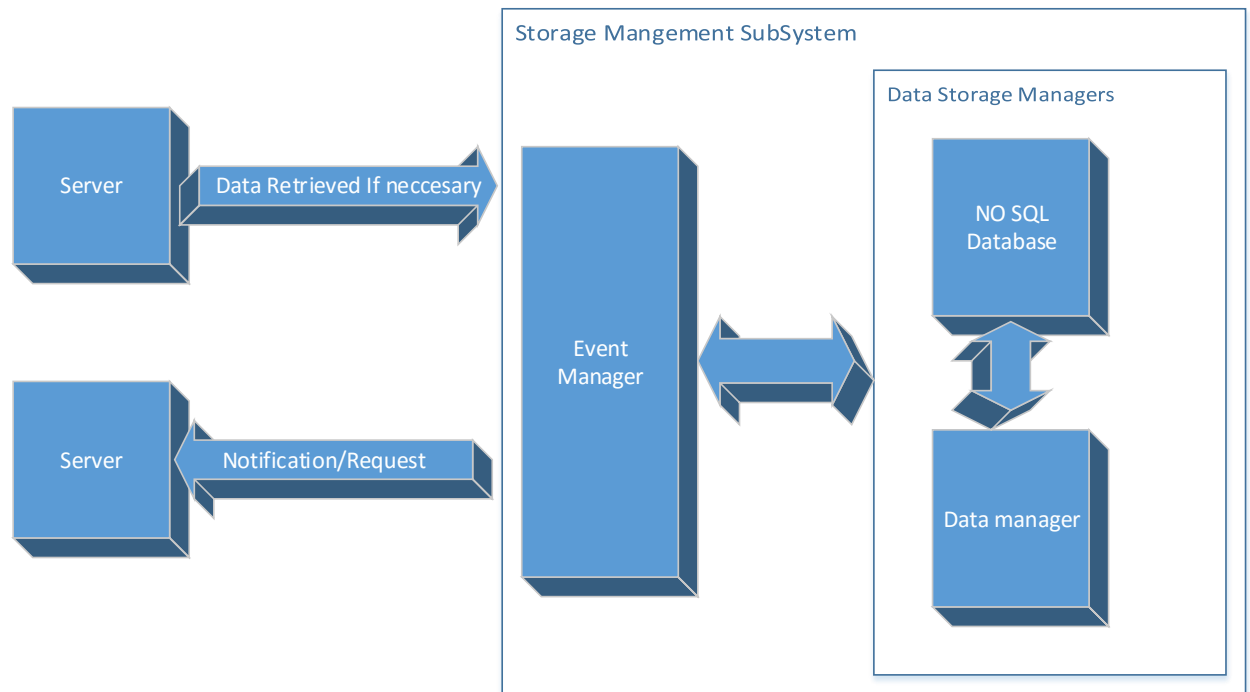
Test team makes sure that the project met the entire specified requirement by means of testing with the test drivers. Test Team tests the developer's source code with the test drivers provided by the testers. SCF is really useful for this purpose, as it provides comprehensive summary of testing any developer's code.

2. Storage Management Subsystem

2.1 Architecture

The Storage management subsystem provides a number of services that includes storing and managing. The SMS (Storage Management Sub System) has different responsibilities in different servers. All responsibilities of SMS with respect to server are explained in other sections (Section 2). SMS is the core part of Software Collaboration Federation. SMS consists of:

Database, Data manager, Event Manager.



Data Storage Managers

It is composed of Data manager and database.

Data Manager is responsible for managing the data from the database. It is also the responsibility of the database to notify the event manager to send the notification to appropriate persons or to appropriate servers. Data Managers use the NoSQL DB to store, edit, and retrieve data needed by a specific application, e.g., Repository code management. They make a generically useful database into a specific tool for some set of tasks.

Data storage managers are code that wrap the NoSqlDb, Comm and Data Manager to make it easy for the Repository, Test harness, Clients, and Collab servers to do their jobs. We can think of the managers as Facades that make it easy to use these facilities.

There are variety of things that are handled by data managers like:

- 1 The Data Manager will look into the request from the client. The request is different for different server and clients. The request from client will be in XML. In that case, the data manager has to parse XML and then collect necessary data from No SQL database. The data managers is divided into data manager parser, data controller and XML message producer. However, additional packages can be added to the data manager. Data manger parser parses any incoming new request and converts into readable format by data controller. The data controller can be taken as task manager which contains a set of commands for different servers. The data controller assigns the tasks based on the command tag in XML request. The XML for all servers will be described later on. When the task is performed. Then, it is necessary to convert the result in standard xml string format which is done by the XML message producer.
3. It may be possible that event manager receives data in byte format and that data may be build image or test code. Then there is a need of data contract and our data contract looks like this.

```
public class Message
{
    [DataMember]
    public string fromUrl { get; set; }
    [DataMember]
    public string toUrl { get; set; }
    [DataMember]
    public string content { get; set; }
    [DataMember]
    public byte data { get; set; }
}
```

The above is the general outline of data contract. It varies differently for different servers.

There is no need to provide appropriate value to each and every field of our data contract. However, there are some guidelines that must be followed so that user or server or client can get the desired result

1. fromURL – specifies the source URL that consists of its IP address and port number.
2. toURL – specifies the destination URL that consists of IP address and port number.
3. Content – The data manager will automatically predict that the XML is passed as string and if there is not string passed. Then, it is not possible for data manager to understand the request. However, this field van be empty.

There is also a format of XML that all clients/servers has to follow.

```
<message>
<command>request1</command>
<body>...</body>
</message>
```

Here the command tells the data manager that what operation it has to do.
For each server, there are fixed number of commands that it should handle and every client and server must be aware about it.

Body is a dynamic so it will be discussed specifically in all the servers.

Body can be empty.

4. Data – This field is used if the server is sending build images and test code to the SMS.

If the test code or build image is big. Then, the data can be passed in chunks.

And in the content portion, the server can specify whether this is the beginning or ending or progress of the file. Everything must be present in the form of XML string.

Event Manager

Event Managers are responsible to send notification to the necessary users for notifications and to servers in order to notify or get data or to send data. Event Managers use the communication system to provide a routing mechanism for each of the SCF servers to notify. The event Manager is divided into event sender and event receiver. The event receiver receives all the requests from the user and place it inside the message queue to be picked by the data manager. The event sender receive all the response from the data manager and place it in queue until the message is sent via communication channel.

There are following responsibilities of event manager:

1. It is the responsibility of event managers to notify the users and the collaboration server if the code is committed.
2. The event managers is also responsible to send code for committing, to send build image and to send responses.

2.2 Responsibilities of SMS

The following are the highlights of responsibilities of SMS:-

1. The SMS in Collaboration server is used to store concept documents and workflows. It is the SMS which communicated with repository server to give data e.g. Product status etc. to the collaboration service. So, SMS can also provide web services. The Architect or team lead can allocate resources to work packages, schedule work packages.
2. There are various specifications and design documents that are used to create the outline of the project. Those documents can be used by client if client wants to see the report or any kind of document. Those documents can be stored in SMS of collaboration server and retrieved later on upon request. This retrieval and storage of data is managed by data manager.
3. Developers from different teams write their own code. So, there is need to check in and check out of code. This thing is managed by SMS of repository server. Developers can

contact repository server and then SMS is used to store the information about code committing and various files that are checked in. It is helpful in version control. After the code has been checked out. The event manager of SMS is used to send the notification to all the persons associated with the code.

4. The client or developer can build the code they want to test. That is done by build server. Again, SMS is used here to store the build information and to handle the request messages of the client. Upon receiving the request, the data manager of SMS inside build server first checks build image that might resides in the database of SMS. If it's not there then Build Server requests the Repository Server for providing the Test Suites along with the Source code. Repository Server will serve the source code, test suit and XML mapped file to the Build Server. Then Build Server compiles this source code to generate build image and forwards it to the Test Harness Server. So, it is the indirect responsibility of SMS to execute unit tests, integration tests, regression tests, performance tests, stress tests, and acceptance tests.
5. SMS can be used for reporting progress, key events, and failures, adding meeting and retrieving meeting schedules.

2.3 Storage of Data Structures for Servers using SMS

Storage of data structures in Database of SMS for collaboration server

The collaboration server receives data structures in the form of xml string .The information is stored as a key value. The DB Element in collaboration server for workpackage looks like below:

```
public string task { get; set; }           // metadata |
public string description { get; set; }     // metadata |
public string person { get; set; }         // metadata |
public DateTime timeStamp { get; set; }    // metadata value
public List<string> dependency { get; set; } // metadata |
public List<string> payload { get; set; }   // data |
```

Suppose the server receives add work package request. Then, key will contains the work package information such as work package name will be the key, the task inside the db element will contain the task. The description contains the information about work package. If the work package has any dependency then it will be in dependency column. Payload will contain extra information needed for allocating tasks.

DBElement in collaboration server for meeting

```
public string name { get; set; }           // metadata |
public string description { get; set; }
public int meetingLength { get; set; }     // metadata |
public string meetingOwner { get; set; }
public DateTime meetingDayTime { get; set; } // metadata value
public List<Key> personInvolved { get; set; } // metadata |
public List<string> payload { get; set; }   // data |
```

Similarly, like that our DB will store data for meeting information and any additional information will be stored in payload like whether to show code or not, whether to show action item or not otherwise the payload will remain empty.

The data manager reads the data differently from database for different tasks.

Storage of data structures in Database of SMS for repository server and build server

```

public string name { get; set; }           // metadata   |
public string description { get; set; }     // metadata   |
public DateTime timeStamp { get; set; }     // metadata   value
public List<Key> dependency { get; set; }   // metadata   |
public string filepath { get; set; }        // data       |

```

The structure of DB Element for repository structure will remain but the name of variable will become different for better understanding. While checking in the code into repository. The key will be the name of the package and name will be the person who is checking in the code. The description element contains the information about the package. Dependency element description will be same as above. The filepath element will contain the path of the server where the file is stored.

In case of build server, all the fields will remain same, the key will be build number.

Storage of data structures in Database of SMS for test harness server

```

public string environment { get; set; }     // metadata   |
public string version { get; set; }         // metadata   |
public string updateTo { get; set; }        // metadata   |
public string descr { get; set; }           // metadata   |
public DateTime testcaseInitiation { get; set; }
public DateTime testcaseEndTime { get; set; } // metadata   value
public List<Key> packages { get; set; }     // metadata   |
public List<string> payload { get; set; }   // data       |

```

In the test harness server, the test config number will be the key. The environment have the environment number. The description will have the description about the config number because the tool like JUNIT will contain the same config number and the test cases that are predefined by the testers. The packages will contain the list of packages that corresponds to that test case. The element packages will not have any functional advantage. It is just for the information of the client.

2.4 Critical Issues

Issues regarding Eventual Consistency

The Scheduler persists the database into XML after sometime. If the Exec Package wants to access the data before the scheduler schedules the dictionary data into XML, then it will lead to problem of eventual consistency.

Solution: The problem of eventual consistency can be solved if data is first searched from DB engine or in memory and if the data is not present in in memory. Then, it will search from persist XML.

Fault Tolerance

Customer data is very important. Due to failure or any exceptional condition, there is a chance that data can be lost. So, if one data shard is lost, others may not be of much significance.

Solution: To overcome this problem, our software system must have data replication mechanism which can store the replica of the data in persist engine and logs the location of data into logger module. The logger module can be considered the metadata that contains information about replications of data.

3. Collaboration Server

3.1 Summary

Collaboration Server consists of SMS in which there are data and event managers.

The task of event manager is to give notification to the corresponding person/team when the event has occurred.

The task of data manager is store data. It stores management information such as work packages, schedules and virtual displays. Work packages has to be at least one tangible thing. Here one tangible thing means at least one thing either test document, implementation, unit test that the tem can say that it's complete. The Team lead along with software architect may review the work packages of all the teams.

There is also a dependency diagram which shows the dependency of all the works in a project and also intra- dependency of modules.

There are multiple users of the server: Customers or Client, Developers, Architects, Tester.

Many different teams are there in the project and there may be dependency of a person in a team with another person. This dependency tree can cause serious issues and cause lag which will be discussed in critical issues.

3.2 Organizing Principles

The important tasks of the team like adding the meeting schedules, interaction with the team. Also, assigning the team member their responsibilities with the work packages is an important responsibility which is handled by collaboration server.

3.3 Users and Uses

This collaboration server is used by:

1. Team Members: Team members will get notification about the code check in/ check out and about the meeting schedules.
2. Team Lead/Project Manager/Software Architect: Team Lead and project manager uses collaboration server to schedule meetings and make the chart of monthly meeting schedule. Team leaders of different teams can make their own schedules and can also organize intra meetings such as meeting between testing and development team.
3. Software Architect/ Manager can retrieve product status, documents and code.
4. Customers or Clients: The client can use this server to get details of the build whether it is passed or failed. Also, client can make meeting schedules. And view reports.
5. To briefly highlight uses, the Collaboration server is used to display virtual meeting, used by clients to show reports, used by different members of the team to add/view workflow packages, to schedule meetings, to view reports.

3.4 Responsibilities

The Responsibilities of Collaboration Server are:

1. It stores management information such as work packages, schedules and virtual displays.
2. To provide notification to the concerned person/team members if any change occurs in repository.
3. Retrieve product status, documents, code which can be used by Team Lead/ Manager.
4. Run collaboration tools and view reports which can be viewed by client.

3.5 Data Structure, Data Contract, Commands and XML Format for Collaboration Server

The data contract is already explained in Section 2.1. This subsection gives the information about XML and data contract specific to collaboration server.

So, the content section in data contract consist of XML as described in section 2.1 has a field command and body. We will discuss more about body and command here.

The commands will be interpreted by the data manager of SMS.

Data Structure

The SMS handles variety of data structure in collaboration server. It includes list of strings, strings which is wrapped in XML string format. All the requests received by the collaboration server is in XML. So, it is necessary to parse XML.

Data Contract

```
[ServiceContract(Namespace = "Collab Server")]
public interface ICollabService
{
    [OperationContract]
    string retrieveMeetingDetails(string metaData);
    [OperationContract]
    bool addMeetingDetails(string metaData);
    [OperationContract]
    bool addWorkPackage(string metaData);
    [OperationContract]
    bool retrieveWorkpackage(string metaData);

    [OperationContract]
    bool configVds(string msg);
}

[MessageContract]
public class CollabTasks
{
    [MessageBodyMember]
    public string fromUrl { get; set; }

    [MessageBodyMember]
    public string toUrl { get; set; }
}
```

```
[MessageBodyMember]  
public string content { get; set; }  
  
}
```

The message contract of VDS will remain same

Commands

1. add_workpackage – This command if the client wants to add Work Package.
2. retrieve_workpackage – This command if the client wants to retrieve Work Package.
3. add_meeting_schedule – This command if the client wants to add meeting schedule.
4. retrieve_meeting_schedule – This command if the client wants to retrieve meeting schedule.
5. add_codecommit – This command is used to add code commit details.
6. retrieve_codecommit - This command is used to retrieve code commit details.
7. retrieve_runtools - This command is used to run collaboration tools.
8. retrieve_documents – this code send the documents and product status to collaboration server.
9. clear_logs – it clear all the logs stored by event logger
10. get_logs – It is used to get the all the logs.

XML Format for Meeting

```
<message>  
<command>add_meeting_schedule</command>  
<body>  
<meetingDay>01/01/2016</meetingDay>  
<meetingTime>13:00</meetingTime>  
<meetingLength>3HR</meetingLength>  
<employees>6<employees>  
<showActionItem>true<showActionItem>  
<codeWindow>true<codeWindow>  
<productivityWindow>>false<productivityWindow>  
<personInvolved>  
  <person>abc</person>  
  <person>xyz</person>  
<meetingOwner>ojuneja<meetingOwner>  
</body>  
</message>
```

XML Format for WorkPackageAllocation

```
<message>  
<command>add_workpackage</command>  
<body>  
<task>abctask</task>
```

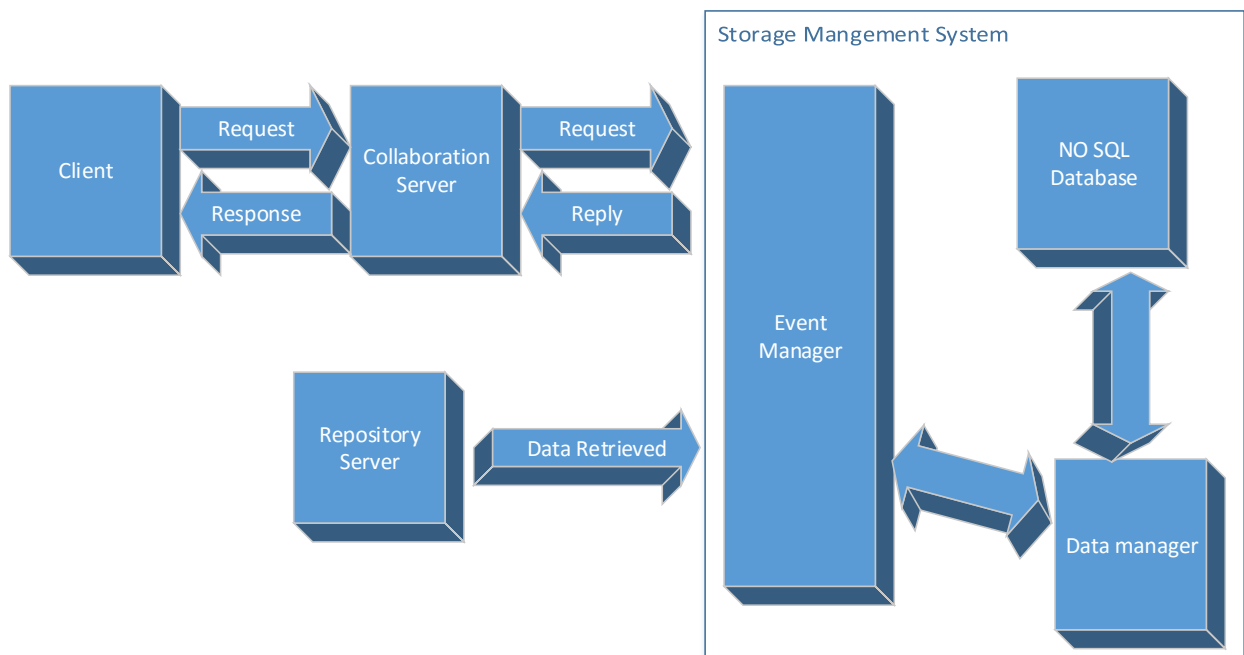
```
<descr>decription<descr>  
<package>e1.cs<package>  
<people>e2.cs<people>  
</body>  
</message>
```

Explanation:

The command section is already explained above. In the body section, the xml format varies. For example: to run the tools like VDS, the xml will contain the information about start and end of meeting. The address of remote locations, the information about the team leads or managers that should be involved in the meeting.

To retrieve or add workflow packages, the XML must contain the dependency information and the work package which is assigned to each team member and its dependency. The XML standard cannot be fixed for even this work package information as it will contain a hierarchical tree with lots of things involved.

3.6 Architecture



Explanation of Architecture and Activities of Collaboration Server

In the above Diagram SMS is a part of Collaboration server but to show interactions, I have made them separate to explain it properly.

First of all, repository server always send data to SMS of collaboration server and SMS then store the data into its database.

The client interacts with Collaboration server to get product status/documents/view reports. Client can use any GUI or run tools like VDS (virtual display system which will be explained in detail in later section) to interact with server. To make it simple, I did not included in the architecture diagram.

Role of SMS in Collaboration Server

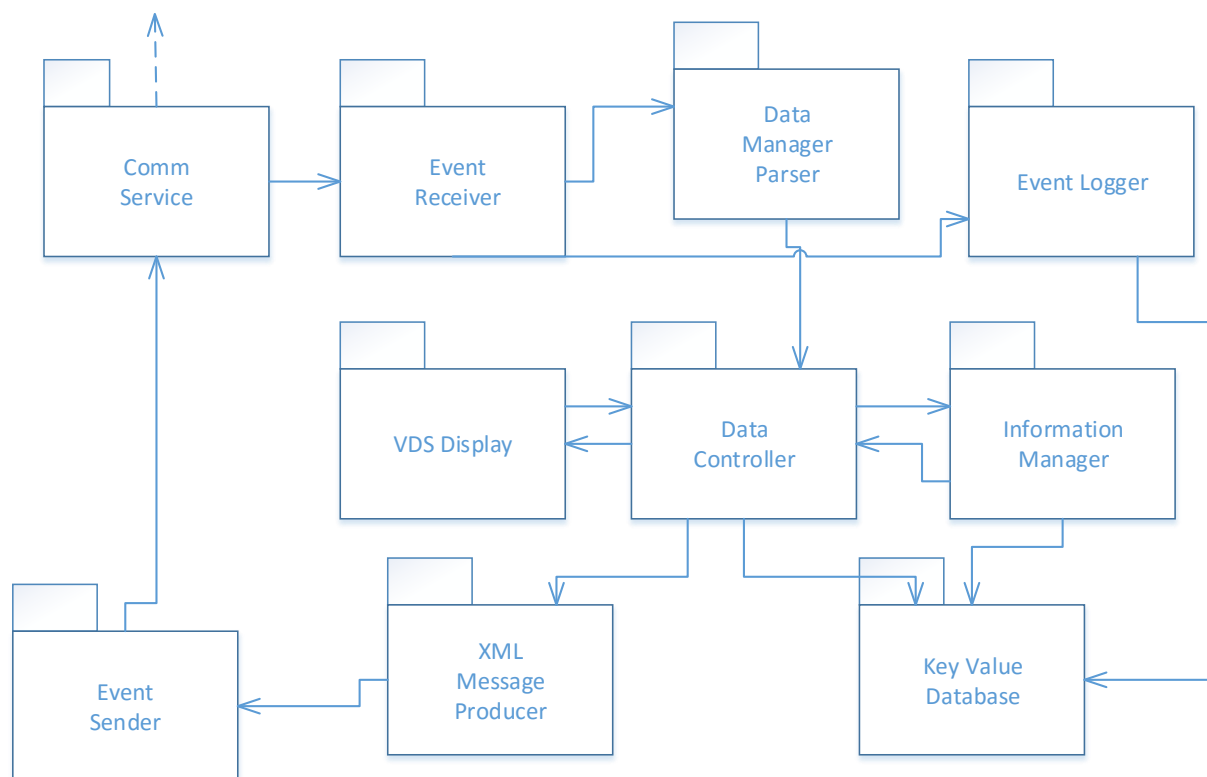
Note that to make meeting schedule and to make workflow or to retrieve data, the data manager which is present inside SMS will use NoSQL Database of SMS.

Whenever any meeting schedule is added or any workflow is added, then the data manager also interacts with the event manager to send notifications to the necessary users.

In case of meeting schedule, the necessary users are involved in the meeting and also sent to the team which is dependent on it.

In case of Workflow package added, the notification is sent by event manager to all the members of team.

Package Diagram



Explanation

DATA Storage MANGER Components – Data manager parser, data controller, XML message producer, Key Value Database

Event Manager Components- Event Sender, Event receiver, Comm Service

1. The Event Receiver of Event Handler receives the message through the communication service. The Event receiver has a queue to maintain the messages. The Event receiver then passes the message to the Data Manager parser.
2. Data Manager parser parses the message and send the message details like command and body of the message to data controller. Data manager parser is a part of Data manager.
3. The Data Controller can be considered as the most critical part of data manager. There are multiple commands and data controller assign the task with respect to the command. The data controller assigns all the information tasks(addition or retrieval or editing) to the information manager.
4. The information manager all these tasks. If the command is to add/retrieve any kind of data in the data base. Then, it contacts the No SQL database to add/retrieve the data. When the data is retrieved then, it passes the message to data controller again which passes the information to XML Message producer.
5. If there is VDS display request, then it contacts the VDS to do start the meeting and display and VDS do its task and send all the information to the data controller. The data controller send the message to XML Message producer as well as store the details into the database.
6. The task of data controller is not to format message, it will just pass the message to message parser. The main function of data controller is to assign the tasks according to the command.
7. XML Message producer then wraps the message into proper XML string format and then place the message into the sender queue of event sender. Event Sender is a part of Event handler.
8. The Event sender takes the message from its queue and then send it to the client via communication service. The VDS is also a part of Collaboration server but its complete explanation is discussed later in this section. Data handlers and Event Handlers described in this diagram is a part of Storage Management Subsystem.

Activity Diagram

The Collaboration server can be used by multiple users. The request to collaboration server are divided into 3 types:

First of all the message is parsed by using the message parser and then the type is determined.

Type 1 is the meeting request: Meetings can be added, initiated. All the requests which is going to be added into the database is done by the information manager which will add the meeting

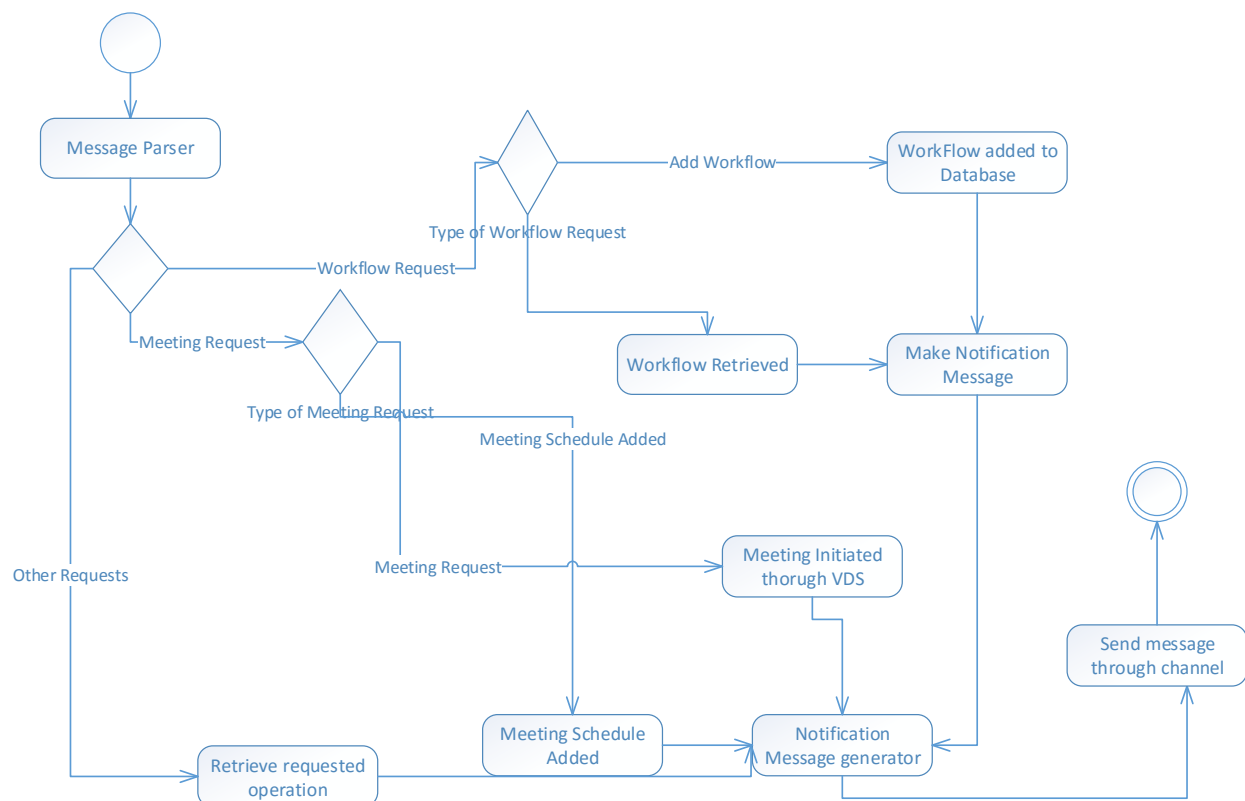
schedule into the database. The meeting request can also be initiated from the VDS and the details can be added into the database.

Type 2 is the workflow request: It is necessary to prepare adequate workflow. The workflow can be stored and retrieved. The workflow package info is added into the database through the information manager and retrieval process is also carried out through information manager. There is an additional activity which will analyze the dependencies in workflow and store the information related to it into the database.

Type 3 are other requests like retrieval of code, product status etc.

All the information which are retrieved is formatted into proper XML string and placed in sender queue and then the notification is sent to client via event handler.

Note that the data stored is managed by data manager and type of requests are triggered by event manager which are part of SMS which is sitting inside collaboration server.



3.7 Role of Agents in Collaboration Server

A software agent is a computer program that acts for a user or other program in a relationship of agency: an agreement to act on one's behalf. Such "action on behalf of" implies the authority to decide which, if any, action is appropriate. They are small, customizable units that can be used to personalize the system. It consists of:

- Trigger – can be a time, an event from another tool, or a manual trigger. In collaboration server. When any event is generated like adding work package, retrieving work package, scheduling meeting then the event is triggered.
- Tool – It is a tool that runs when the trigger is fired. So, whenever the triggered is

fired. The event logger logs the event and stores the log information in separate XML in No SQL database. The logs related to meeting are only logged into the database and the logs will automatically get cleared after some amount of time. The time will be set by the administrator . Infact, clearing the log is another agent activity.

3.8 Critical Issues

1. Inefficient Workflow Management: If one person is dependent on any other person for its work then it's called lag or locking of resources. It cause late completion of project or deadline may be passed which is not favored by managers. So, as result it is bad performance by team with respect to client and with respect to management.
Solution: This issue can be handled by having appointing single software architect for a single team and carefully making workflow with respect to other projects if any dependencies exist.
2. Responsible for Delivery: If any member of the team is not able to deliver its work on time then whole team cannot be blamed for it. Instead the person should be blamed so that it will not cause conflict in the team. This is a serious issue if the workflow is not efficient or any team member is incompetent in delivering its part. So, this issue should be considered seriously and should be handled as described above.
3. Meeting Overlap: It may happen that single person is having two meetings on same day and same time. It may create problem for the team members.
Solution: There must be a meeting checker functionality while allocating the meetings that it should not clash. If the two meetings clash, the additional software should tell before even creating the meeting schedule.

3.8 Virtual Display System

VDS servers as the main interaction for the team. In the virtual display system, the remote team members which are involved in the meeting can see documents required for meeting along with the remote team. Also, these teams can write on the virtual display system. The display system can sense the pencil writing on the screen. can also sense the speech and convert it from text to speech. These all the things happens with the help of low power radio transmitters and receivers.

3.8.1 Virtual Display Functionality

1. Creation of UML style drawings

- a. A user draws a free hand sketch of a small component consisting of perhaps a dozen lines or so, and the display transcribes that into a VISO-like element by matching the sketch with a series of component templates.
- b. Perhaps the user will select the strokes that make up a sketch and tap on the drawing surface near the sketch to activate this conversion.
- c. These components can be dragged around the drawing surface and joined with connectors.
- d. The user can also drag a component from a toolbox to the drawing surface.
- e. A drawing can be saved, retrieved, and edited at any time.

f. Text notes can be attached in callout style boxes to any element of the drawing, and can be hidden or made visible by a pen-based gesture command.

2. Special purpose charts and visuals, based on user-defined templates can be created, placed anywhere on the drawing surface, edited, saved, and retrieved at any time. These charts show information about a current software development project in ways that are briefly described below, and which you will explore as part of this project.

3. Free-hand sketches of components, diagrams, and charts may be created anywhere on the drawing surface, edited, saved, and retrieved for later use. Free-hand sketches of components may be componentized later by matching to predefined templates, as in #1, above. Evidently this will work well only for relatively simple sketches.

4. HTML documents can be created, saved, and retrieved from storage and placed anywhere on the drawing surface. Text from the document can be selected with a pen and edited with voice commands.

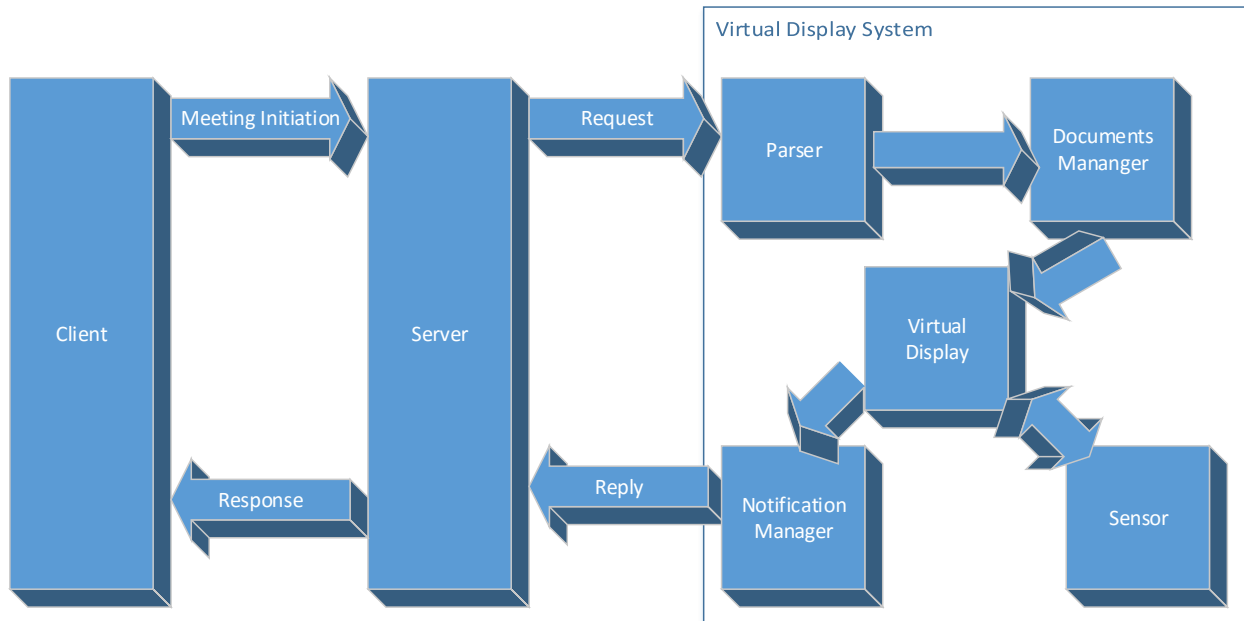
5. Webcam images of collaborators can be placed in resizable windows anywhere on the display surface, and linked through Voice over IP.

6. Text messaging that serves as short term storage of communication between participants, but which can be saved to Collaboration Server storage.

[References: Functionalities are taken from:

<http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/projects/Pr5F12.pdf>

3.8.2 Architecture



Explanation

The virtual display system is a part of collaboration server. I have made it separate in order to explain VDS. The client initiates the request for meeting in virtual display system. The virtual display system has its own parser in which it receives the request for meeting. The parser reads the message which the virtual display system can understand. There will be some documents needed for the meeting and according to the parsed message, it forwards the information to the document manager where multiple documents are shared with the virtual display. The sensor is a different part of the Virtual Display system which manages what managers or team members want to share at runtime. There are tools, pencils from which information can be shared. When the connection is setup and all the preprocessing has been done, then the notification manager tells the SMS which is present inside the server to send notification to the client that the meeting request has been processed and the meeting is initiated.

4. Repository Server

In order to understand the tasks and the activities carried out at the Repository Server, the Repository Server subsystem is divided into the following module.

4.1. Repository Server Overview

The repository server is the heart of Software Collaboration Federation where the entire data of the project is located. It provides the dependency storage of certified code, documents and provides production analysis tools. Also, whenever any code is committed then the repository server stores that kind of data into separate XML and notifies the collaboration

server about updation. It also supplies code to test harness. Client can also check in, check out code and can also run analysis tools and view reports. The repository server contains SMS which is divided into data managers and event managers. The data managers parse the message and do the operation like check in check out, versioning, store modules. The event managers

Main functions of Repository Server include:

1. It stores modules as a collection of packages
2. Supports check-in, check-out, extraction.
4. Allows extraction of modules for the Client and maintains a record of checked-out modules.
5. Manages metadata of all modules that are checked-in and checked-out.
6. Provides code and mapper xml to the build server.
7. It provides package information to the collaboration server.

4.2 Organizing Principle

The Repository server is the core of software collaboration federation. There is a strong need to check in and check out the code written by developers. Also, it act as backbone for the other servers. As all other servers whether directly or indirectly getting data from this server.

4.3 Users and Uses

This repository server is used by:

1. Team Members: Team members can check in and checkout the code.
2. Team Lead/Project Manager/Software Architect: Team Lead and project manager uses repository server to view reports and do versioning.
3. Collaboration server uses repository server to get commit information, product status and documents.
4. Customers or Clients: The Client can use the server to view the commit details or reports.
5. Build Server uses repository server to get the code and XML mapping file required to build for the test harness.

4.4 Data Structure, Data Contract, Commands and XML Format

The data contract is already explained in Section 2.1. This subsection gives the information about XML and data contract specific to repository server. So, the content section in data contract consist of XML as described in section 2.1 has a field command and body. We will discuss more about body and command here.

Data Structure

The SMS handles variety of data structure in repository server. It includes list of strings, strings which is wrapped in XML string format and bytes of data (code files) which is received in data

section of data contract. All the requests received by the repository server is in XML. So, it is necessary to parse XML and receive bytes of data.

Data Contract

```
[ServiceContract(Namespace = "Repository Server")]
public interface RepoService
{
    [OperationContract]
    byte checkOut(string fileName);
    [OperationContract]
    bool checkIn(string filename, string metaData);
    [OperationContract]
    string retrieveCheckInDetails(string fileName);
    [OperationContract]
    byte sendCodeToBuildServer(string metaData);
}

[MessageContract]
public class RepoTasks
{
    [MessageBodyMember]
    public string fromUrl { get; set; }

    [MessageBodyMember]
    public string toUrl { get; set; }

    [MessageBodyMember]
    public string content { get; set; }

    [MessageBodyMember]
    public byte data { get; set; }
}
```

Commands:

1. checkin_code : It will add the code to the cache manager until the whole code is not received.
2. checkout_code: The code will be fetched from repository and sent client can edit then.
3. view_report: The client can view the report for any checkin and checkout
4. send_code: the code is sent to the build server.
5. retrieve_codecommit - This code committing report is sent to the collaboration server.
6. retrieve_documents – This code send the documents and product status to collaboration server.
7. newbuild – this command along with mapper xml and code is sent to build server to notify that raw materials has arrived to make new build.
8. clear_logs – it clear all the logs stored by event logger
9. get_logs – It is used to get the all the logs.

XML Format:

The XML format for the repository server is different from other servers. The XML format is:

```
<message>
<command>checkin_code</command>
<body>
<code>true</code>
<beg>true</beg>
<mid>>false</mid>
<end>true</end>
<test>true</test>
<dependency>
    <file>f1.cs</file>
    <file>f2.cs</file>
    <file>f3.cs</file>
</dependency>
<size>380000</size>
<owner>Ojas Juneja</owner>
</body>
</message>
```

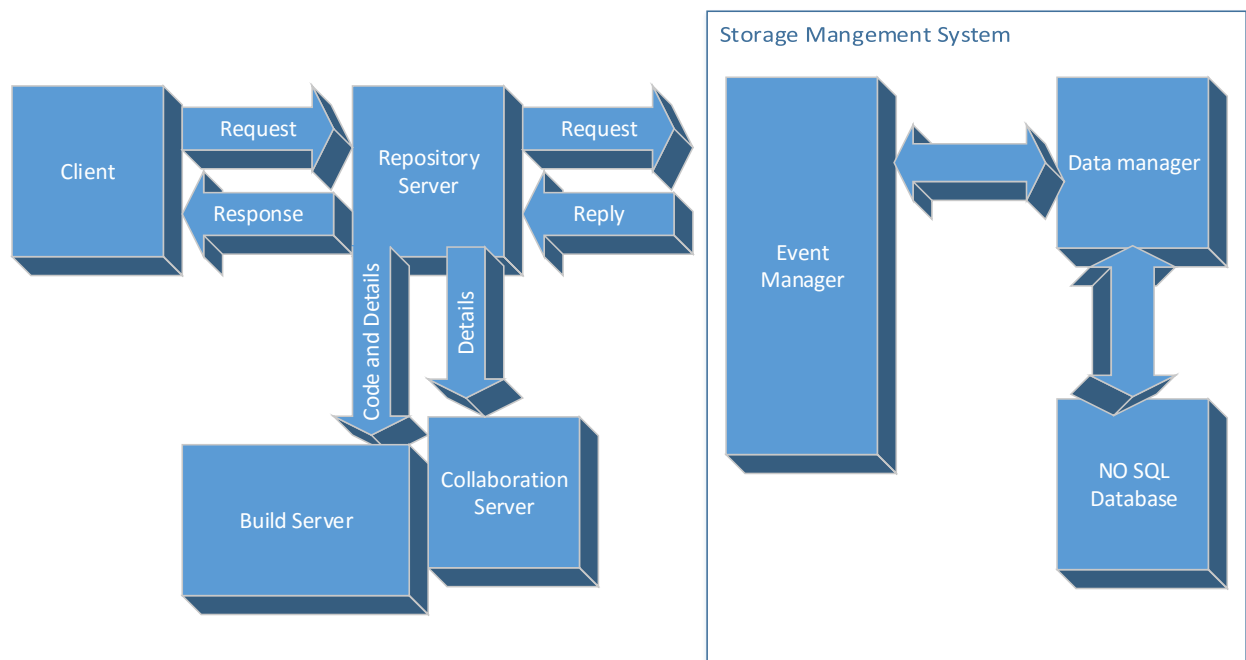
Explanation

The dependency tag contains what are the dependencies when the file is checked in and checked out. The size is the size of file being sent. Owner tag shows the owner of the file. If the code field is set to true that means the code is checking in or checking out or it may be sending to the build server. If the code flag is not set as true then there is no meaning of test flag and XML can be sent as needed.

There may be a need to transfer the code in chunks. Then the beg tag is set to true if first chunk is sent and mid tag is middle of code and end tag is last chunk. If all the tag is set to false, then there is no code and data manager will ignore the code chunk if the code is sent along with XML. However, if the beg and end and/or mid tag is set to true then data manager will assume that there is only one chunk of code. However, if the first chunk is sent and the mid or/and end tag is true. Then, the data manager will not be able to predict which chunk it is and it will be ignored. So it should be handled carefully. Below is the table which explains it clearly.

State	beg	mid	end	Status
1	TRUE	TRUE	TRUE	Accepted as last chunk
2	TRUE	FALSE	FALSE	Accepted as first chunk
3	TRUE	FALSE	TRUE	Accepted as last chunk
4	TRUE	FALSE	TRUE	Accepted as last chunk
5	FALSE	TRUE	FALSE	Accepted as mid chunk if the first chunk is there
6	FALSE	FALSE	TRUE	Accepted as last chunk if the first or/and mid chunk is there
7	FALSE	TRUE	TRUE	Accepted as last chunk if the first or/and mid chunk is there
8	TRUE	TRUE	FALSE	Accepted as mid chunk if the first or/and mid chunk is there

4.5 Architecture



Explanation

The repository server passes the code to the build server if build server needs it. Also, after doing checkout it updates the information in the collaboration server also.

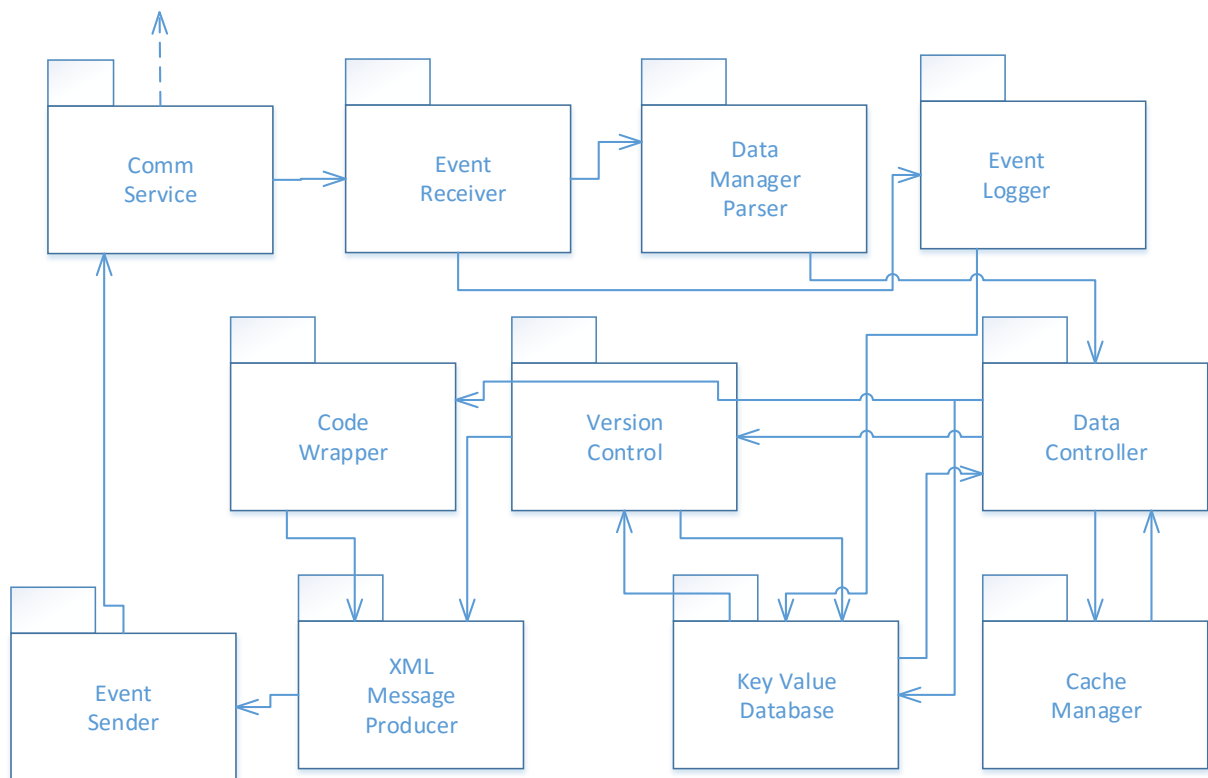
Role of SMS in Repository Server

The SMS (Storage management Subsystem is a part of Repository Server). I have made it separate in order to explain the diagram. The client send the checking and check out request in XML format. The event manager receives the request and then passes to the data manager. The data manger interprets the message and check out the code and do the versioning. The client

can ask for reports and can also run analysis tools. The build server can ask the Repository server for the build. The event manager receives request and asks the data manager for the code and stores the dependency relationships and build information into XML string and then supplies the code to build server through event manager.

Whenever the code is checked out, the event manager sends the notification along with the information in XML string to the collaboration server. The information is the documents, code and the product status whenever any change occurs in the database of repository server. The collaboration server sometimes needs this to review its workflow packages and to share documents in meetings.

Package Diagram

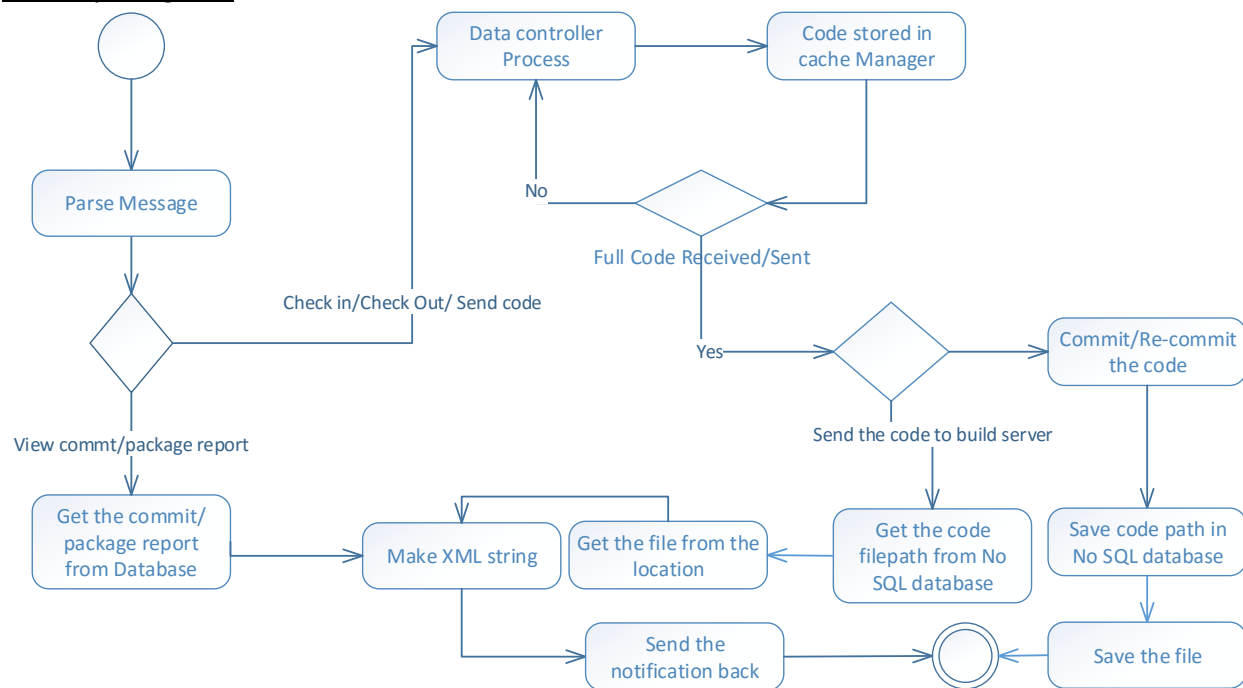


Explanation

DATA Storage MANGER Components – Data manager parser, data controller, XML message producer, Key Value Database, Cache Manager, Code Wrapper

Event Manager Components- Event Sender, Event receiver, Comm Service

1. The Event Receiver of Event Handler receives the message through the communication service. Now the message can be received from the build server or from the client. All the messages are placed in queue of the event receiver. The Event receiver then passes the message to the Data Manager parse
2. Data Manager parser parses the message and send the message details like command and body of the message to data controller. Data manager parser is a part of Data manager.
3. There are multiple commands and data controller initiate the task with respect to the command. If the command is to check in/checkout the code and if the code is too big. Then, it may be transferred in chunks of data as bytes. The data controller will receive the xml string and check beg, mid, end flag which is already explained in section 3.4. If more code is yet to come then, the code is stored in cache manager for temporary storage and in the end it is sent to the version control which will be explained in more detail in this section.
4. The version control software manages check in and check out and also maintain versioning of the code. The code file which will be received by the repository server is saved in a separate location. In our No sql database, the path to that file will be stored.
5. Similar procedure is followed if the code is needed by the build server but there is one more step. The code wrapper is there which will attach the information about the code which will be required to build the code. That information is the dependency information of all the packages. There is also a format for code wrapper XML which will be discussed while discussing build server.
6. The client may want the information about code checkin and checkout and versioning, then the data controller contacts with the key value database and get the info about versioning and pass it to the code wrapper. Note that, before passing the information to the XML message producer, it is necessary to do the proper formatting of information and code wrapper does that. XML message producer will wrap the message properly into xml string and then send it to the destination via event sender which uses the com service. The Event sender also maintains a queue to send the messages.

Activity DiagramExplanation

When the message is received by the event manager of the SMS. It is sent to message parser of data manager. The data manger parses the message and classifies it according to the type.

1. If the message is checkin/checkout/send then data controller which is a part of data manager performs its action by appropriately placing it inside the cache manager until it receives all the bytes from the client. The above step in case of checkin/checkout process but if the message is of type send code. Then, it means that build server wants the code to generate its build in that action there is need to further classify the message. If the message is checkin or checkout then the code is committed and if the message is send code to build server. Then it get the path to that file from no SQL database and from that path the file will be fetched. Our No SQL database maintains a separate XML for this which contains all the information about the data which is committed. The information is extracted and XML mapper is generated along with the code and message is send back finally.

When the code is committed then the code is finalized and after committing is done. Then the server will persist the code in another asynchronous thread. The location of that code file is stored in No SQL database and the file is stored at that particular path. After the committing of code is done. Then the notification is generated and added the sender queue of event handler to send it across network to the client.

2. If the message is to view/commit report. Then, the necessary information is extracted from the database and then then sent back. Note that there is XML parser which makes the message string which is an XML string.

The activity diagram hides all the details regarding the receiving of code which is completely explained in section 3.4

4.6 Role of Agents in Repository Server

Agents are small, customizable units that can be used to personalize the system. It consists of:

- Trigger – can be a time, an event from another tool, or a manual trigger. In repository server, it will be triggered whenever there is check-in, checkout or request from build server to send code.
- Tool – It is a tool that runs when the trigger is fired. So, whenever the triggered is fired. The event logger logs the event in the No SQL Database. Note that the logs can fill the database with too much data. So, it will be cleared after some amount of time as fixed by the architect. The check-in , checkout and send code logs are recorded in the same XML File.

4.7 Critical Issues

1. Concurrent File commit

As the checkout is carried out in server, it may happens that both user take the updated version of the file. After taking the updated version of the file, it may happen that multiple users can check-in into the file and then send check in request to server at the same time.

Solution: The problem can be solved by providing the locks on the file. If the client sends the checkout request. Then, the server will not allow any other client to checkout that file until it has been checked in. So, only one client can work on each file at a particular time.

2. Unauthorized Repository Check-In

The code stored in each file can have many dependencies. So, fault in one file may bring down the complete software. There are many new hires in the company working on the same project and on a particular functionality. The problem may arise if they commit the file with a buggy code or they checkout the file and corrupt the file and check-in again

Solution: it is the responsibility of the team leader or the team manager to proper authenticate the client before checking in and checking out. For this, the repository should ask for the username and password before check-in. The username and password can be employee id and password and only team leader or team manager can grant the right to have access to the repository.

3. Server down while check-In

The client can check-in any number of files to the repository and sometimes there is huge load on server. So, there is a limit that x number of bytes should be transmitted in

one go. It may happen that the server is down in between and the other bytes cannot be sent. But if the file is committed, then whole software can crash.

Solution: We are maintaining tags like the beg tag, the medium tag and the end tag. The check –in will never happen if the server will never encounter the end tag and if server becomes down. The client have to recheck-in the file again.

5. Test Harness Server

5.1 Introduction

Test harness server is used for testing developer's code by running pre-defined test cases. It accepts build files and libraries as provided by build server. Test Harness is a means to show whether the build is succeeded or not. If all the test cases on a particular module is passed, then the build is succeeded otherwise not. Test harness also has its own SCS in which the XML template is decided. The XML template supplied to test harness Server contains the XML mapping information which is required to build. The test cases will run either by client request or user request. It contains the test run configurations and requires the build image from the build server. The client can see the test configuration. However, it is the task of tester to put the test configuration.

Test Harness server has the following responsibilities:

1. It tests the build image from the build server to run the code.
2. Saves the test results, summary and logs.
3. Provides notification to the person who ran the test configuration or developer if stated.
4. Test harness server can do regression testing, performance testing.
5. To update the libraries if requested present in test harness server for the testing tools and inform the client.

5.2 Organizing Principles

Test Harness server is an important part of the Software Collaboration Federation. It is required to test the quality of code and to be sure that all the requirements are fulfilled as stated by the client.

5.3 User and Uses

This test harness server is used by:

1. Team Members: Team members can test their own code and testers can test the code.
2. Team Lead/Project Manager/Software Architect: Team Lead and project manager uses test harness server to view test reports.
3. Customers or Clients: The Client can use the server to view the quality of code and to see the test report and the progress of the project.
4. It saves the time of tester by automating a lot of things.

5.4 Data Structure, Data Contract, Commands and XML Format

The data contract is already explained in Section 2.1. This subsection gives the information about XML and data contract specific to test harness server. So, the content section in data contract consist of XML as described in section 2.1 has a field command and body. We will discuss more about body and command here.

Data Structure

The SMS handles variety of data structure in test harness server. It includes list of strings, strings which is wrapped in XML string format. All the requests received by the test harness server is in XML. So, it is necessary to parse XML. Also, the build image is received by test harness server is in bytes.

Data Contract

```
[ServiceContract(Namespace = "TestHarness Server")]
public interface TestHarnessService
{
    [OperationContract]
    string test(string metaData);
    [OperationContract]
    string getConfigNumberDetails(string metaData);
    [OperationContract]
    string getBuildNumberDetails(string buildNumber);
    [OperationContract]
    string getBuildNumbers();
}

[MessageContract]
public class TestHarnessTasks
{
    [MessageBodyMember]
    public string fromUrl { get; set; }

    [MessageBodyMember]
    public string toUrl { get; set; }

    [MessageBodyMember]
    public string content { get; set; }

    [MessageBodyMember]
    public byte data { get; set; }
}
```

Commands:

1. test_config – It run the test cases for the given configuration
2. info_build – It fetches the information of the particular build
3. info_run_last – It fetches the report of the last run
4. info_build_all – It fetches the information of all the builds.

5. Info_run_all – It fetches the report of all runs.
6. Info_run_progress – it fetches the information of all the runs that are going in server.
7. Info_run_testconfig – it fetches the information and report of a particular run.
8. fetch_config_numbers – it fetches the information of all config numbers
9. update_tools – it updates the version of the tool or update the library.
10. build – it sends the message to make the build image.
11. retrieve_build – it indicates that the build image is coming along with the XML.
12. clear_logs – it clear all the logs stored by event logger
13. get_logs – It is used to get the all the logs.

XML Format

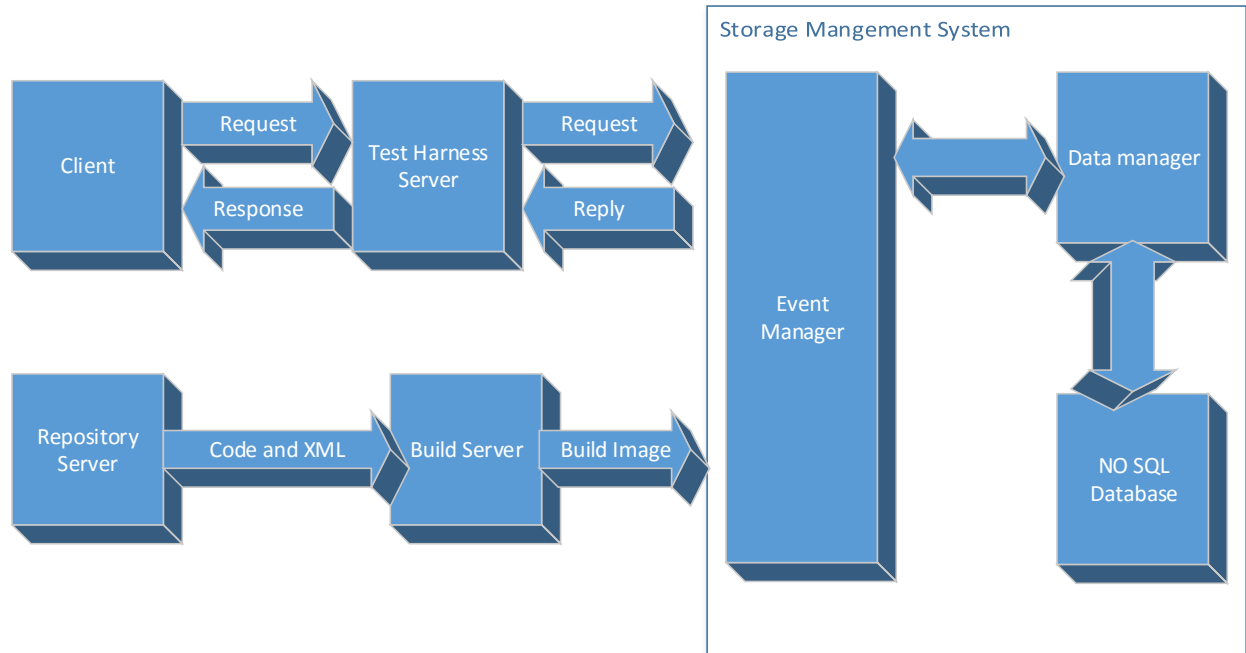
```
<message>
<command>test_config</command>
<body>
<build>null</build>
<beg>true</beg>
<mid>false</mid>
<end>true</end>
<config>
<tool>JUNIT</tool>
<version>1.0.0</version>
<updateto><updateto>
<confignumber>12ed3</confignumber>
<environment>qa</environment>
<browser>chrome</browser>
</config>
...
</body>
```

The XML format for test harness server is different than other servers. The change is in the tag build in which client can specify the build. There is also config tag according to which client should specify the environment in environment tag and type of browser in browser tag if there is web testing. There is separate config number for each run and client can use the config number and run the exact configuration without mentioning the details in the config file. The config number is automatically generated by test harness server and client can leave the config number blank if it don't want to run the older test config. However, if the client specifies the config number then all the configuration is simply ignored even if the config number is wrong. If the client don't know the config number but wants to run the old config then it use the above commands to get the test config details.

There is an update to tag below the version tag of too. This tag when not left empty will update the tools with its libraries for the particular test case or set of test cases. If the update to tag is same as version flag, the, vaue in updateto flag is simply ignored. The beg, mid and end tag is

the beginning of build image, build image in progress and last chunk of build image. The complete and detailed explanation of beg, mid, end can be found in section 3.4.

5.5 Architecture



Explanation

Test harness server can be taken as a standard to measure the correctness of code and completion of requirements. The request come from the client to do a task. There can be multiple requests as mentioned in the 4.4 section.

I have made the Test harness Server separate from the SMS in order to explain it clearly. SMS is a part of test harness server.

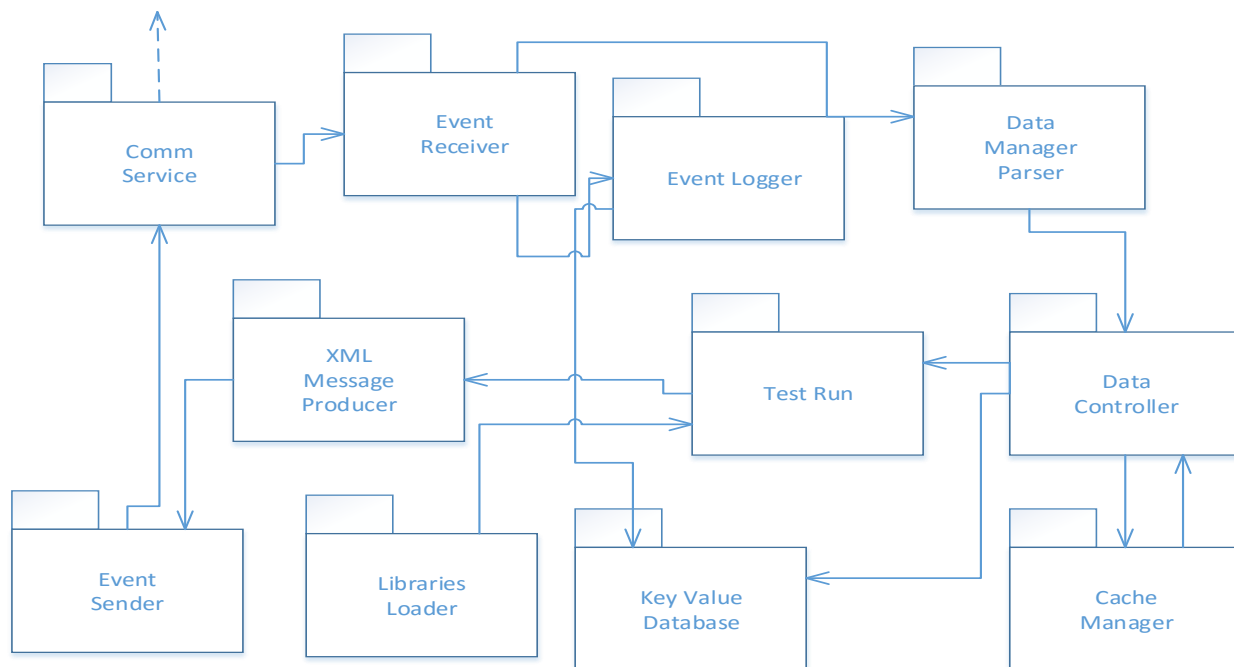
Role of SMS in Test Harness Server

The event manager receives the request through communication service and transfer it to data manger. It is the task of data manager to parse the message and do the task. The client can ask for the previous builds or previous run configuration. If that is the case then it takes the data from database and then make the XML string and then send it to event manager to send for delivery. If the request is to run the particular test cases by supplying test config info and build is not specified then the data manager ask the build manager to send the latest build and then after getting the build. It runs the build. There are several tools that should be specified by the client in which the test should run. If the build is too huge. Then the build can be send in chunks of data. If the build is not present in the build server then, build server takes the help of repository server to produce the build. The management of test scenarios can be done in Quality Center.

There are number of tools like: JUNIT, QTP, Selenium (for web testing).

After all the test cases is run. Then, the status is passed to the person/client. The event manager is used to do that. The database stores the information of test configuration, build information. However, it never stores the build image as it is the task of build server to give the build image. Data and event manager are both parts of SMS.

Package Diagram



Explanation

Data Storage Manager Components - Data manager parser, data controller, cache manager, key value database, XML message producer.

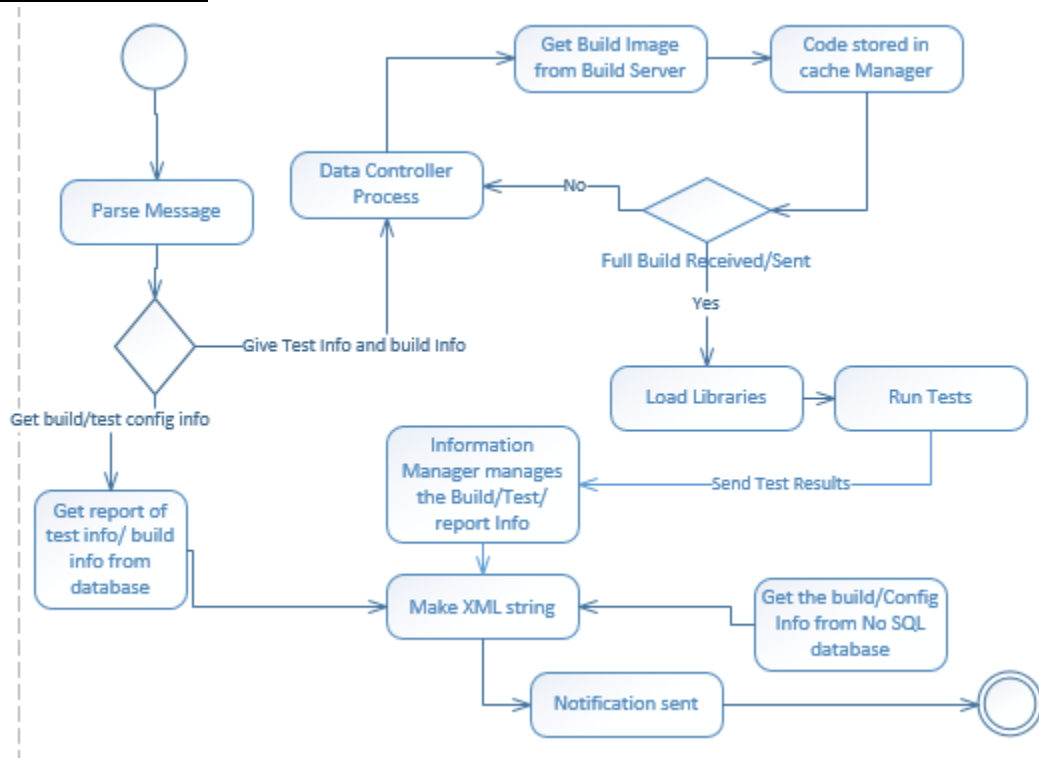
Event Manager Components - Event Sender, Event receiver, Comm Service

The message is received through comm service and placed in queue of the event receiver.

1. The Event receiver then pass the message to data manager parser. The functionality of parser is to parse the message and then send it to the data controller.
2. The work of data controller is to interpret the message according to the commands which are discussed earlier in this section. If the command is to run the test cases, then the test run simply pass the command to build manager which will create the build message and place the message to build the build image in the sender queue of event sender. Note that the test harness will save the configuration which is supplied by the client in temporary storage(cache manager) until the build image corresponding to that is not received and then use the same config to run the test cases.

3. After, the build image is received. Then, data controller receives it until all the chunks are received. The build Image is received in bytes. The cache manager stores the chunks of build and it provides a temporary storage until the complete build is received. Once the complete build is received, then, data controller send the build image along with the test configuration to the test run and the, test run will configure the tools with the help of test libraries loader and then run the test cases.
4. The Libraries loader will load the libraries according to the config file. It is necessary to pass the test config number, tool name and version in which the tests should run otherwise the tests will fail to run. After all the test cases ran, the Test run package pass the information to the XML Message Producer to make the xml string in standard format.
5. The data controller also saves the test configuration and build configuration to the key value database. The task of XML message producer is to make the proper message format and then place it in the queue of event sender. Once the message is placed in queue. Then, the notification is sent to the client.

Activity Diagram



Explanation

The Activity diagram deals with the main activities carried out in test harness Server. The message which is received by Event manager is sent to data manager where it is parsed and then data manager do the operation as mentioned in its command. If the client needs the information about test config or any report or build info then, it is fetched from the database and then data is sent back to the client. Client may request to run test cases or may specify the test build. In the client side, there is test management tool which has the test config number and test cases. In the server side, there is also test management tool which contains the test config number and mapped test cases. The client QC should be synchronized with the server QC. The server QC will produce the config number automatically. The client has to mention test config number in before running test scenario. Whenever the request is sent from client, then the event manager stores the message in queue until data manager fetches it, then, the data manager tells the event manager to contact with the test build server and then until the data controller will receive the build and cache it inside the cache manager until it does not get the complete build. Once it receives the whole build, The data controller send the complete info of test with its build image to run test in the test run package which run the test in tools. The test run will load the libraries necessary to run the test scenarios and then run the test cases. There are different tools like JUNIT, Selenium etc. to run the test. These tools requires different libraries and this activity is done by the library loader. Once all the test cases finished, the test run send the test info and all run info to information manager. The information save the information manager formats and dissects the test information into build info, config info and save it into the database. Note that, all the information is not send to client. Only the test pass the XML message producer. The XML message producer makes the message into the standard format and send it to the queue of the event sender and then send it via communication channel.

5.6 Role of Agents in Test Harness Server

Agents are small, customizable units that can be used to personalize the system. It consists of:

- Trigger – can be a time, an event from another tool, or a manual trigger. In test harness server, it will be triggered whenever there is a request to test the code.
- Tool – It is a tool that runs when the trigger is fired. So, whenever the triggered is fired. The event logger logs the event in the No SQL Database and the logs will be cleared in timely manner after some amount of time.

5.7 Critical Issues

1. Duplicate Test Config Number

Client is passing the test config number to run the test cases. The test config number is an id number which is mapped to a specific set of test cases. If the test config number is duplicate then it may create the problem. The test config number will be duplicated as it

is generated in QC. If the client is offline and the QC will generate the test config number which may clash with any test config number in server.

Solution: This problem is solved if the server has only privilege to generate the test config number. Client can create test cases offline in QC but when it synchronizes the test case with the server then server will generate test config number which is unique and is mapped to single set of test cases.

2. Heavy Load on Server to run Test Suites

There may be multiple clients sending requesting to run test suites. If we allow the test suites to run parallel then, there may be several requests that exceeds the server load. If we allow only one process to run at a time. Then, the test suite which is having high priority may run after a long wait time.

Solution: Threading can be useful but our system should impose some restrictions on the threading. Only particular number of threads should be allowed to run at a time which is within the server performance. However, the unit test should not be run on server. The system can limit that client only run regression test suites on server.

3. Single Test case taking lot of time to run

There may be a case where single test case is taking so much time running indefinitely due to which all the other test case forever remains in queue. The test case cannot only decrease the server performance but also wastes the server time

Solution: An efficient solution is to make the test case fail if it cross the certain amount of time and move with other test case and mention the reason that the test case time run out. In that way the time of tester can be saved. Also, the server performance will not be degraded.

6. Build Server

Build Server is the intermediate server between the Test Harness Server and Repository Server. Build server provides service to compile and build executables for Test harness. Build server uses a build script to build a library or execution image for a specified set of packages. Every test configuration also requires a build before executing tests in the configuration.

Responsibilities

1. Support the testing activity by providing the build image to the test harness server.
2. Contact with the repository server if the build image is not present in the build server.
3. Store the build images with their details in the build server.

6.2 Organizing Principles

Build Server is an important part of the Software Collaboration Federation. It is required to initiate the testing process and plays an important role in the testing of the software. Due to

heavy load on test harness server. It is necessary to make a build server and place it in separate location.

6.3 User and Uses

This Build harness server is used by:

1. Test harness server use the build server to get the build images.
2. Team Members/Architects/Managers can use the build server to get the build info but they cannot build the build image. This is done only by test harness server.

6.4 Data Structure, Data Contract, Commands and XML Format

The data contract is already explained in Section 2.1. This subsection gives the information about XML and data contract specific to build server. So, the content section in data contract consist of XML as described in section 2.1 has a field command and body. We will discuss more about body and command here.

Data Structure

The SMS handles variety of data structure in Build server. It includes list of strings, strings which is wrapped in XML string format. All the requests received by the build server is in XML. So, it is necessary to parse XML. Also, the build image is sent by build server is in bytes.

Data Contract

```
[ServiceContract(Namespace = "Build Server")]
public interface BuildService
{
    [OperationContract]
    bool build(string metaData);
    [OperationContract]
    string buildDetails(string metadata);
}

[MessageContract]
public class BuildServerTasks
{
    [MessageBodyMember]
    public string fromUrl { get; set; }

    [MessageBodyMember]
    public string toUrl { get; set; }

    [MessageBodyMember]
    public string content { get; set; }

    [MessageBodyMember]
    public byte data { get; set; }
}
```

Commands

1. build – This will build the code and prepare the build image
2. build_info – This will get the build info about the particular build.
3. build_info_all -This will get all the build info.

XML Format

The build server receives two type of XML formats:

Format 1:

```
<message>
<command>build</command>
<body>
<buildinfo>build_12gr6</buildinfo>
</body>
</message>
```

Format 2:

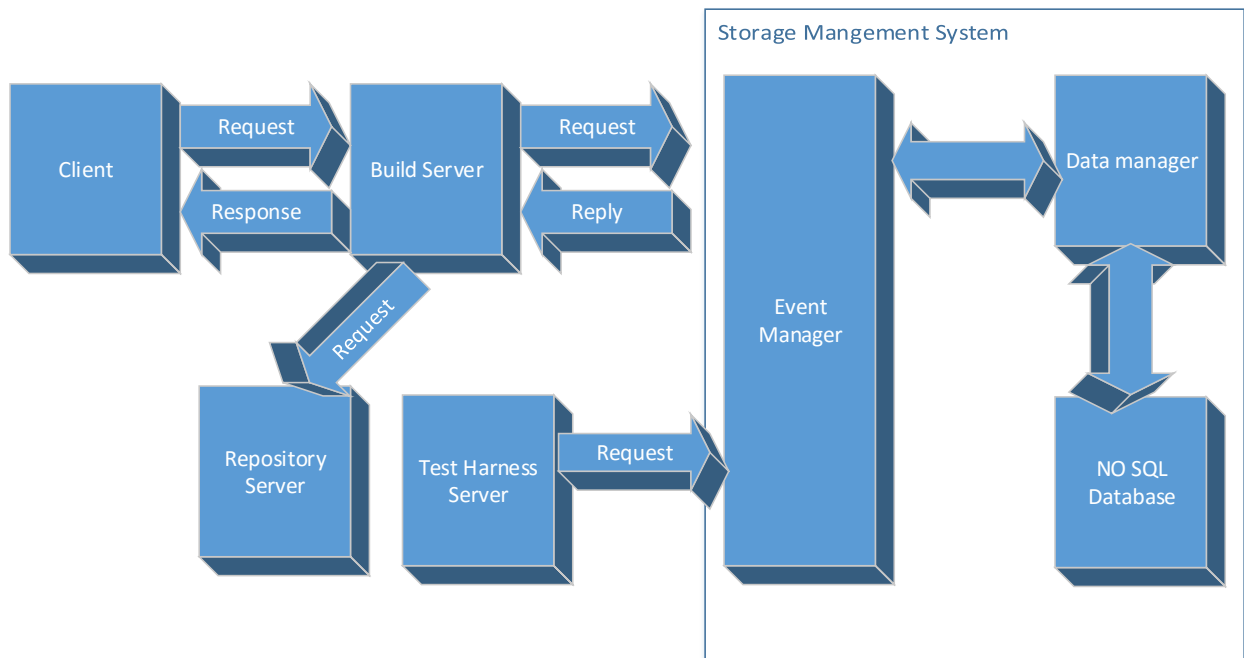
```
<message>
<command>newbuild</command>
<body>
<package>com.cs.utility</package>
  <dependency>
    <file>file1.cs</file>
    <dependency>
      <file>file4.cs</file>
    </dependency>
    <file>file2.cs</file>
    <file>file3.cs</file>
  </dependency>
<size>390000</size>
</body>
</message>
```

Explanation

Build server accepts only format of above type. Here build info tag has the build number. Build info is present in key value database.

The latter format is used when the test harness server is sending the build image and xml mapper file to the build server.

6.5 Architecture



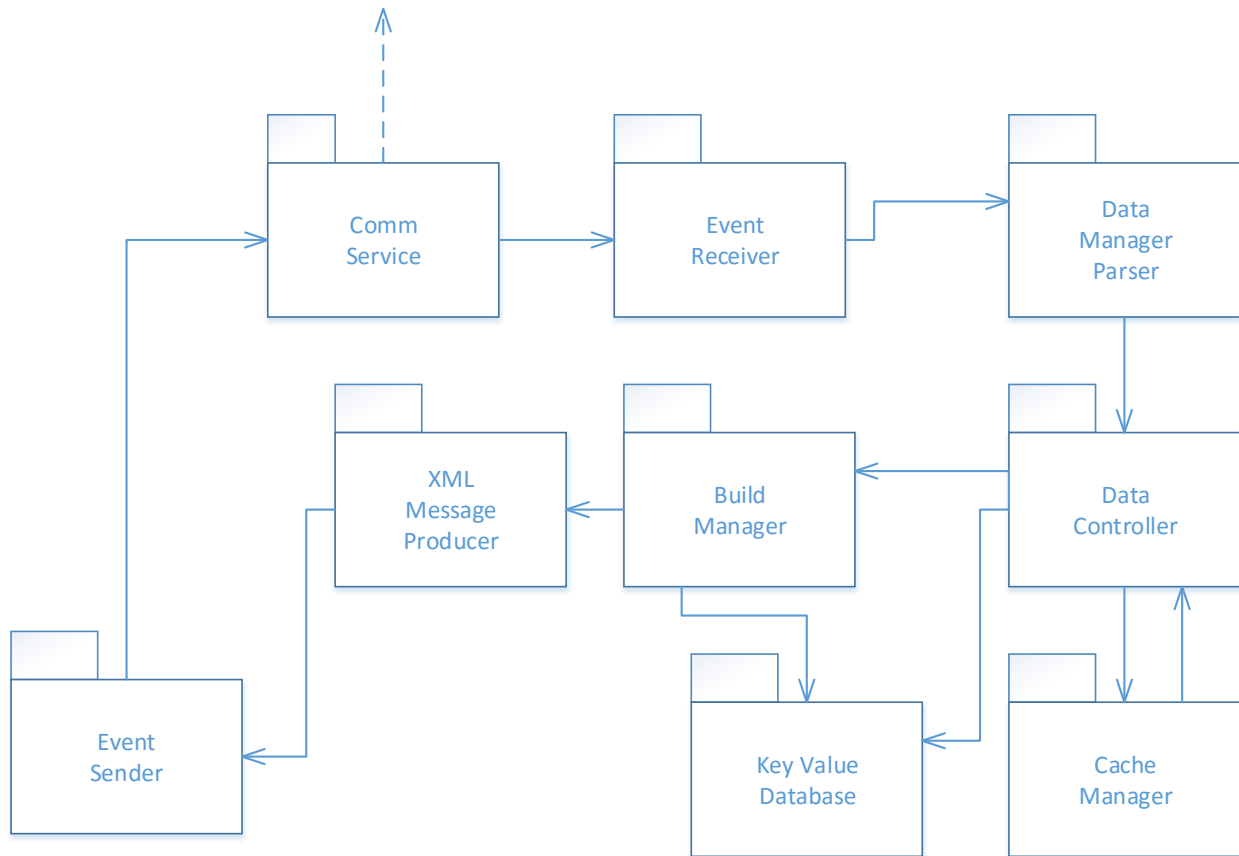
Explanation

Build server receives request from test harness to supply build image. If the test harness server supplies test build number then build server will not build the build image. It will extract the build file from its database and then supplies the build file to the test harness server. If there is a request for new build, then the build server will contact the repository server and repository server will return the necessary code files along with the XML mapper and then build server will build the code and save the build along with the build number into the database.

Role of SMS in build server

The event manager receives the request from the test harness server and send the message to data manager. The data manager manages all the process related to parsing the xml message string and wrapping up of xml string as described above. The event manager then send the notification along with the build file along with the XML mapper to the test harness server. The build server will take the code and mapper xml from the repository server and this sending and receiving communication is handled by the event manager.

Package Diagram



Explanation

Data Storage managers – key value database, data controller, data manager parser, XML Message producer

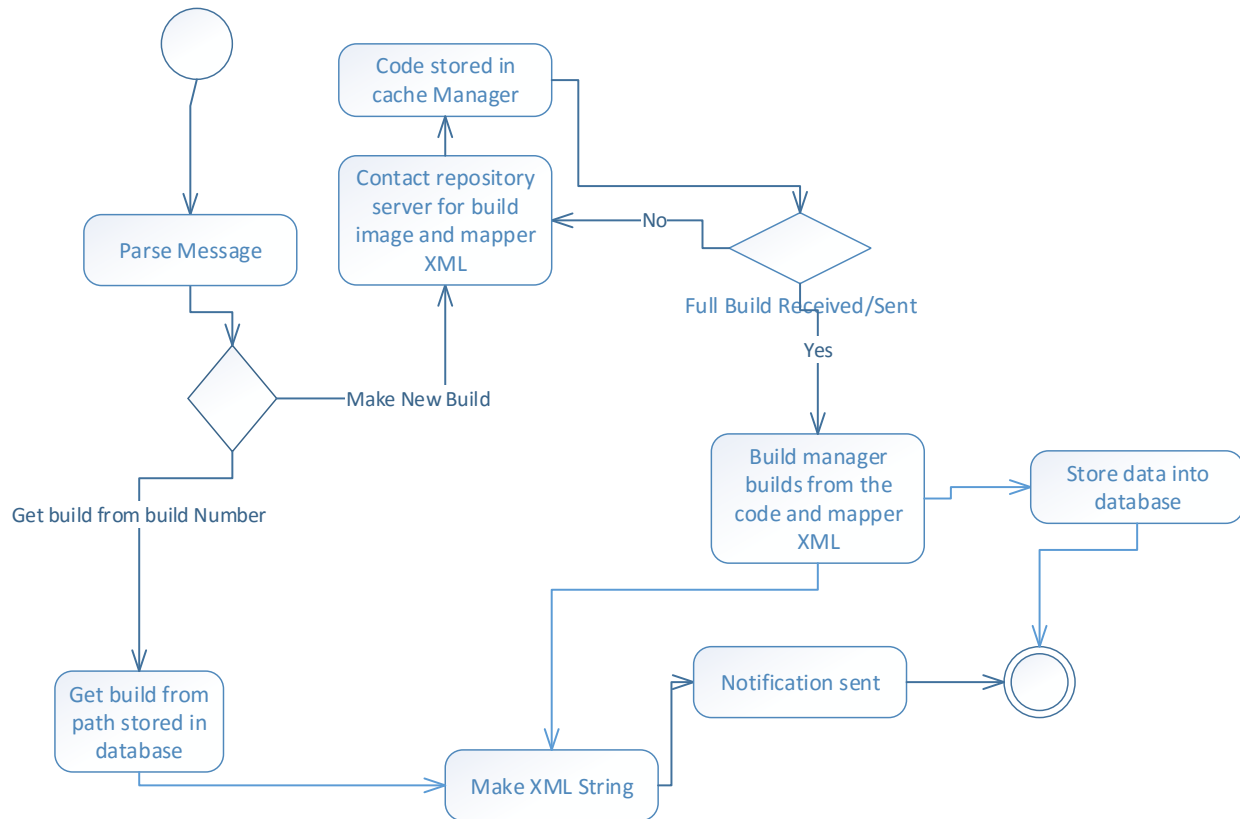
Event Manager – Event Sender, Event receiver, Comm service

1. The Event receiver will receive the message via comm service and place it in the queue of the event receiver. After that message is fetched from the data manager parser and message will be parsed and the command along with the data in the body is sent to the data controller.
2. The data controller will interpret the message according to the command. If the client wants to know the build info then the information is fetched from the database. If the client wants to get the build file with the build number. Then build file is extracted from the file path stored in database with the help of build number.
3. If the client wants to get the new build then the build manger will build the file from the code. The build manager will generate the message and then send it to XML message producer to format the message to send it to repository server. Once, the code along with the mapper XML is received by the event receiver. It will give the file to the data controller and then data controller will call the build manager to build the code and it

will take the help of mapper XML which contains the package dependency information to build the build image from the code.

4. The build image along with the build number is stored in the database by the build manager. The build image transfer the build file as bytes along with the XML and place it into the sender queue of event sender. Then the notification will be send to the test harness via communication channel.

Activity Diagram



Explanation

The message is received by the event handler via communication service from the test harness server or from the client. The SMS present inside event handler parse the request/message and then determines whether the message is to get the old build from database or to get the build information or to make the new build. In case, client wants the old build or build information then it is fetched from the database. In other case, if a new build is requested, then data manager asks the build server to contact repository server to get the code and the mapper XML and when the complete code along with XML is received. Then, the data manager asks the build manager to make the build. Once it is done, the new build information is saved in key value database along with the build number and build file path and place it in the queue of event handler. After that event sender sends the message to the server via communication channel.

7. Client

Based on the tasks and the activities at client's end, the Client subsystem is divided into following modules as described below.

7.1. Client Overview

The Client is the subsystem that invokes all the other subcomponents of the Software Collaboration federation. Client will have GUI through which they can interact with the servers. GUI initiates the actual processing of the our software system. There are different kinds of Client(s).

Project Manager

Team Leader

Developer

Testing Team

Software Architect

All of the users which are mentioned above can do check in and check out to the repository server. The initiation of all the process starts with the client. All the process initiated from the client is mentioned in detail below.

Main features of Client include:

1. Different client will get access to functionalities of Software Collaboration Federation to only predefined functionalities.
2. There are many features available on the client GUI. Many of them include check in and check out, view test reports, view build reports, run collaboration tools.
3. Client can make check in request by adding the code first locally and then checking in to the repository server. Also, it can check out the code from the repository.
4. Client can see the package structure of the system and can view documents that are allowed to view for particular client(s).
5. Project Manager/Team Leader can see the Build report of any module.
6. Client can also start the tools for virtual display system and start meeting and interaction with other parts of team remotely.

7.2 Roles and Responsibilities

Different Roles of the Client are discussed below:

Project Manager/Team Leader

1. Monitors and supervise the development activities going on simultaneously
2. They may check whether the test suites are accurate and fill all the requirements from the test harness server.
3. They may check the code repository and build details from repository server.
4. They can see the package structure and meeting schedules from collaboration server.
5. They assign ownership to various persons involved in the development of the project.

Software Developer

1. They perform the task of building the critical functionalities of the module and then integrating them into the code by using check in and check out functionality from the repository server.
2. They may check that the test suites which are residing inside the test harness server are accurate as per the requirements but cannot edit.
3. They may get package structure of the module and may check their meeting schedules with other team by using the collaboration server.

Testing Team

1. Mainly concern about the testing of the subsystems
2. They may check their meeting schedules with other team by using the collaboration server.
3. They check the test suites and do modifications within the testing suite with the changing requirements from the test harness server.

Software Architect

1. They are mainly concerned with the development of documents for the core functionalities of the system and the basic outline of the system. Also, they develop the interfaces, decide with the team leaders and team that which tools should be appropriate.
2. They are concerned with increasing the functionality of the subsystem and changing or developing the architecture of subsystem.
3. They may check out any module from the Source Code Repository by requesting the Module of the Subsystem.

7.3 Data Structures and XML Format

The data contract is already explained in Section 2.1. This subsection gives the information about XML and data contract specific to build server. So, the content section in data contract consists of XML as described in section 2.1 has a field command and body. We will discuss more about body and command here.

Data Structure

The SMS handles variety of data structure in database. There are different kinds of responses in database. The data structures received are in string, list of strings, bytes etc.

Commands

The Client request XML will contain all the commands as mentioned in the server section.

The response to the client request will be XML or the content in bytes. The XML contains a command which is in the format of response_<request>

XML Format

The XML format will be different for different servers:

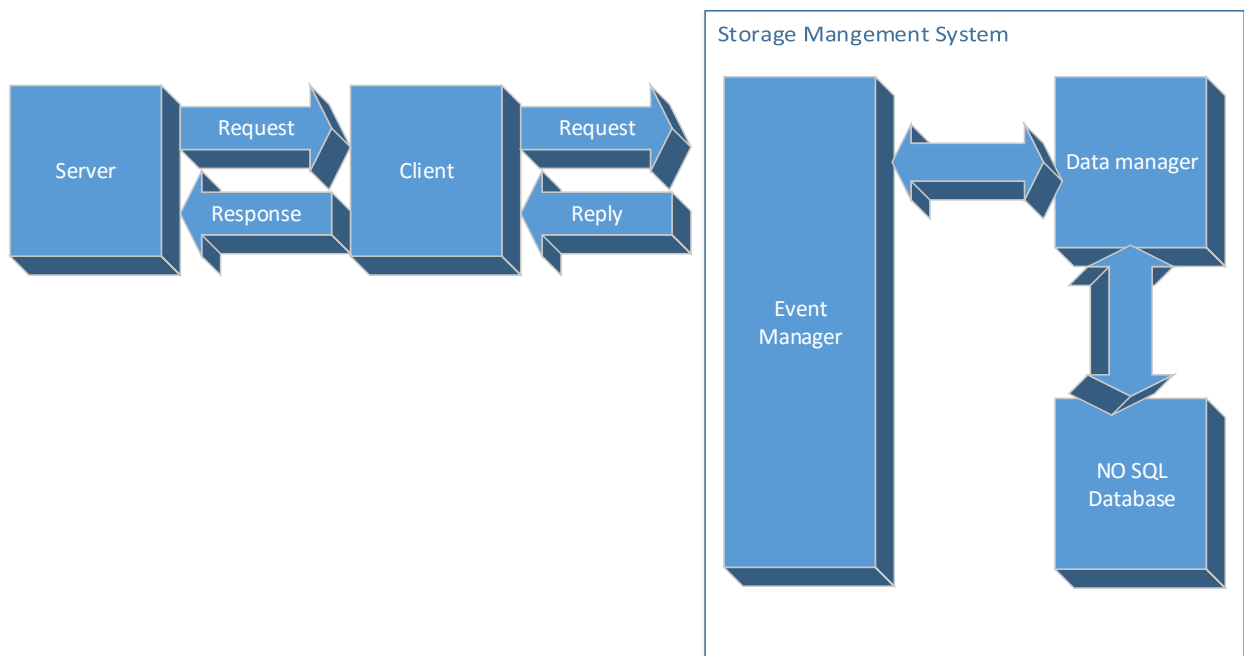
For Repository Server checkout

```
<message>
<command>response_check_out<command>
<body>
<code>true</code>
<beg>true</beg>
<mid>>false</mid>
<end>true</end>
<test>true</test>
...
</body>
</message>
```

Explanation

The xml response may contain bytes of data with the XML. It is the responsibility of XML to provide the information about that. Whenever the file is coming as a response then, there will be mandatory tags that should be attached with XML. The beg tag when set to true shows this is the beginning bytes for the file, mid tag means that file transfer in progress and end tag shows that last bytes has been arrived.

7.3 Architecture



For all servers other than above mentioned

XML

```
<message>
<command>response_xyz</command>
<body>
...
</body>
</message>
```

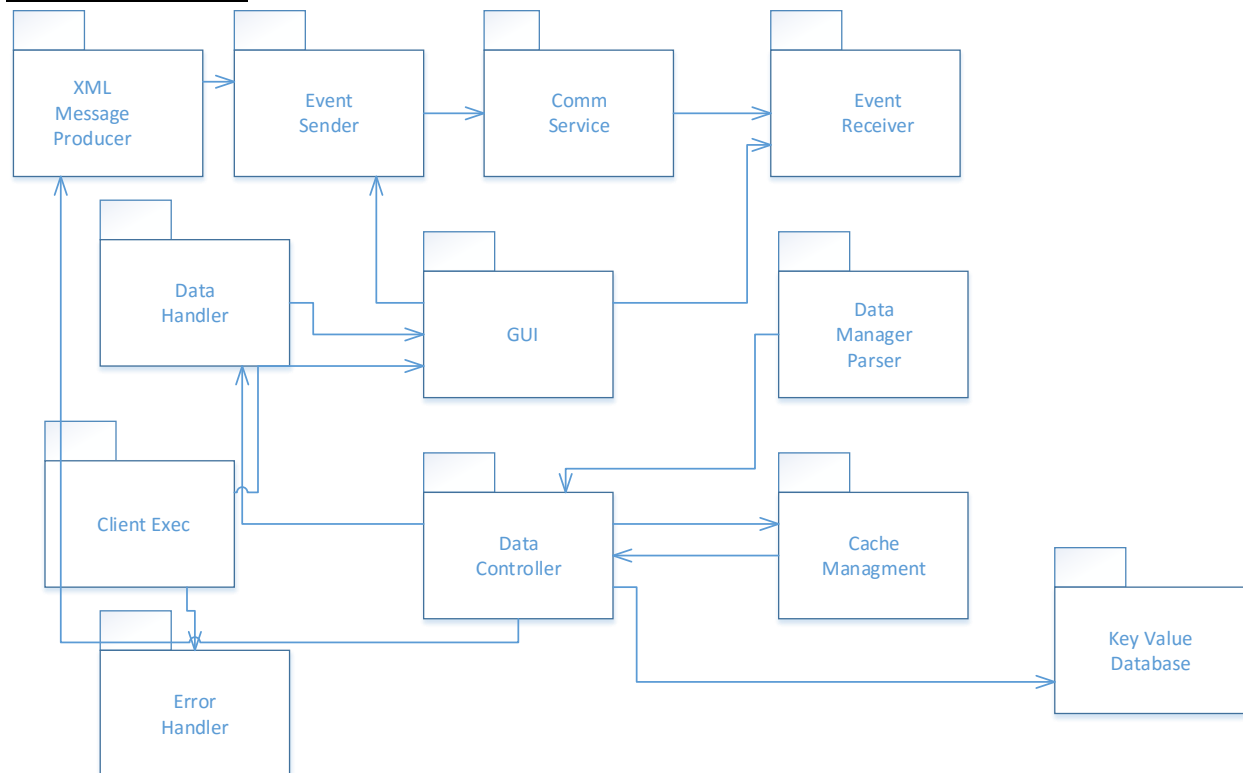
Explanation

There are various clients and can send various requests to various servers. The process of sending request is same for all servers but client has to send requests in different format for different servers which will be explained in server section. SMS is a part of Client and every client may have their own SMS.

Role of SMS in Client

Whenever the client receives response. The event manager get the notification which is type of response. The data manager picks up the message from queue and then parse the message. The parsed message will have command and that way the parser will know how to parse it and where to store it the client local database. There are different kinds of responses in database. It is left to client which thing the client want to persist and which things he/she doesn't want. Data manager also performs the task of sending request. The responsibility of data manager is very critical in client as client can send data or request in different format. The data contract is same as mentioned in section 1.1. The work of event handler is to send the request as soon as the data manager makes the message in proper format.

Package Diagram

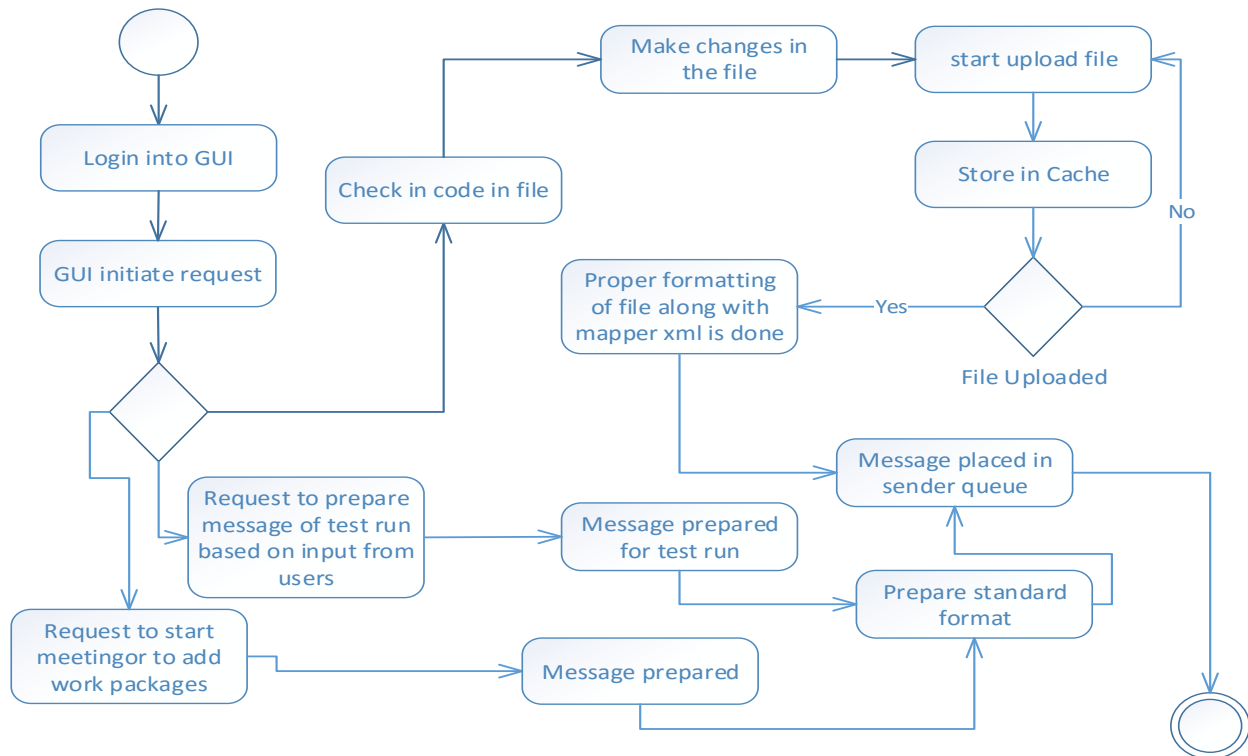


Explanation

The client will receive the response from multiple servers. The event receiver will receive response from the server via communication channel. The data message parser parses the response according to the standard xml format and send the message to data controller. The data controller can be thought as a task manager. Client exec is responsible for starting the GUI and error handler handles the error or any exception occurred while opening GUI.

There can be variety of responses:

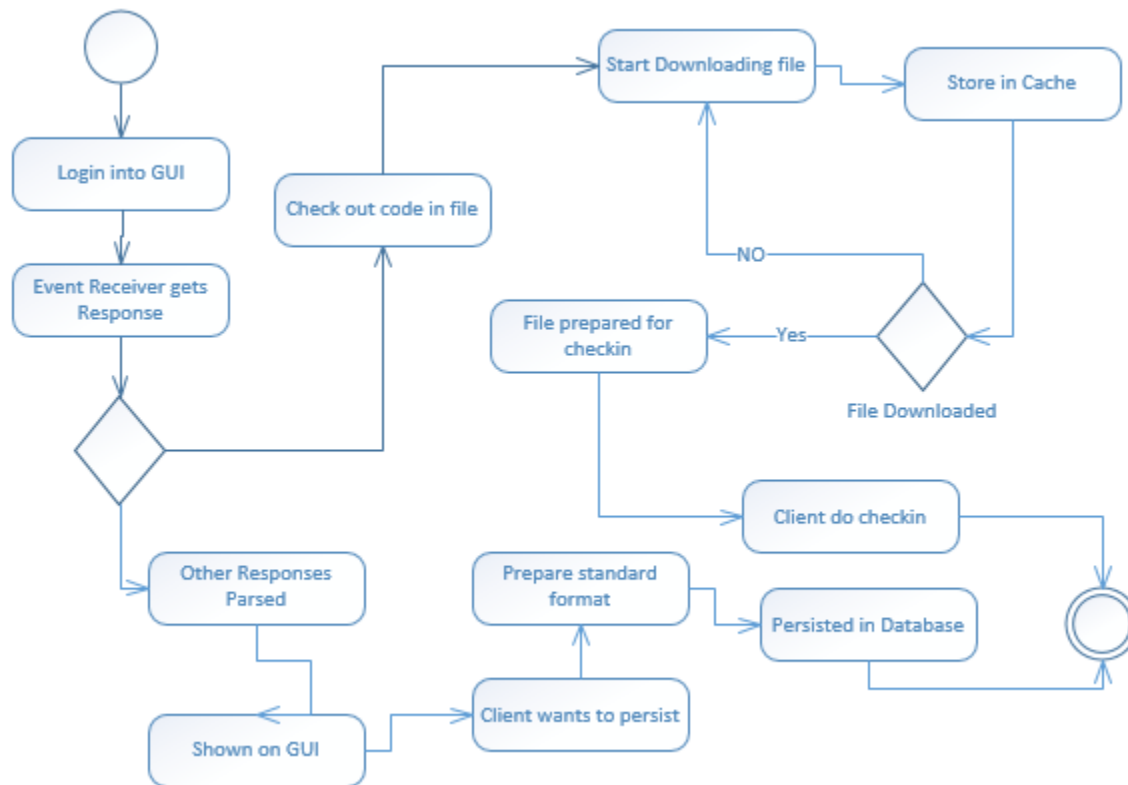
1. If the response is a check in confirmation or build confirmation from the server. Then, the data controller will send all the data to the data handle and then show it to the GUI. Now, it's on the client whether clients wants to save the data in its local or not.
2. If the response is code from the server, then the data handle will wait until all the code is received. Because it receives the code in bytes. So, there is a need for the file caching and the file will be cached until all the code is received by the File Cache.
3. If the client wants to add/retrieve work package, scheduler meeting, get meeting schedule. Then, after getting the details from the GUI, data handler will format the message for data controller. The data controller will send the message to the XML Message producer and it will produce the message in standard XML string format and place it in the Event sender's queue and if the request is sent via communication channel. Then, notification is shown on the client GUI.
4. If the client wants to check out the code, first client needs to take out the code from repository and then client can make changes to the file being edited and after editing client can upload the file. Data controller caches the file until its completely uploaded in the cache manager after the file is uploaded then data controller ask the data handler to make the proper information about the code check-in details . The message is made in proper format by the XML message producer and placed in the sender queue of the event sender. The data portion of the data contract contains the file in bytes. It is sent to the server for committing via communication channel.

Activity Diagram of Message RequestExplanation

The client logs into GUI to send request. Now, there are different privileges for different clients. The client send the request. Now that request are of different types if the client sends request for test run. Then, client needs to fill the details like which test cases to run, which tool should be used, which environment and which browser it has to run test if it's a website testing. The client can also get the test configurations and run any old test configuration by using its configuration number. The proper xml is prepared for that and sent to message parser which prepares xml string according to the data contract. The message producer then place the message into sender queue which can be sent later via communication channel.

Client may want to check in the fresh code or edit the code and then check in. if the client wants to check in the code, then, it first upload the file. Now the file will be uploaded in cache manager first locally and then the message is prepared which contains necessary information about check-in. The message is prepared in standard forma and paced it in sender queue which later will be sent via communication channel.

The client may want to add workflow information, may want to start meeting, may want to retrieve the test report or may want to review meeting schedule. All this request will be prepared with proper details and proper command and then sent to message producer to make it in standard format. That message is then placed in sender queue and then sent to server via communication channel.

Activity Diagram of Message ResponseExplanation

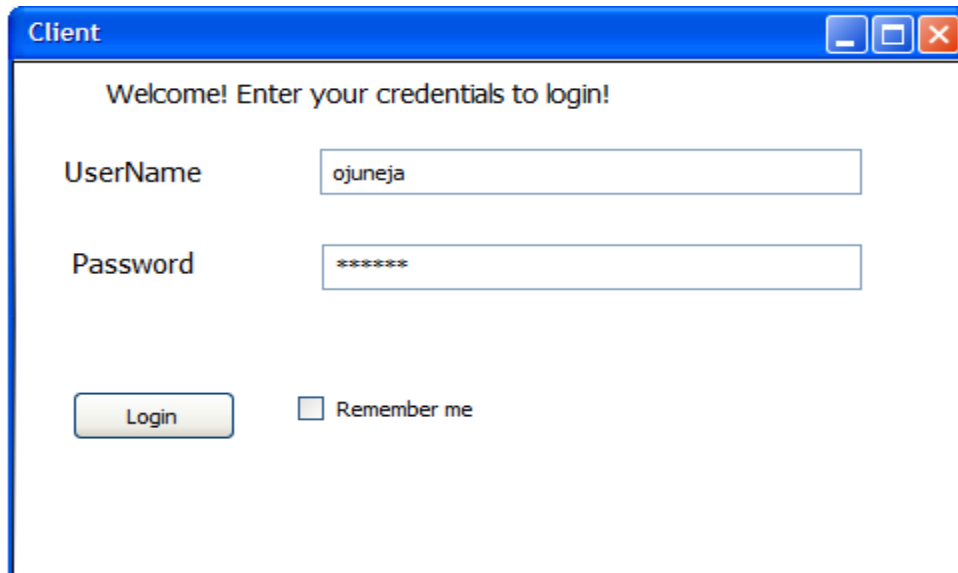
The client gets response which is placed in receiver queue. The response may be xml string or may be xml string accompanied with a file. In case of xml string, the response may of following types: test config file info, build info, work package details, work package added details, meeting details, code checkin successful response. All these messages are parsed first by using the command tag in their xml string file and response is shown on GUI. The client has the facility to save the Responses in its local database in order to maintain a receipt of all these data. If the client wants to persist then the message is first converted to format of key value and then data saved in the meta data section and then it is saved in the database.

If the response is a file that means the client wants to checkout and client has to update the file before checking in. The file is first downloaded and downloaded in cache manager. Then there is versioning tool through which client can update and edit and then check in. I have not explained check in again because its already mentioned earlier.

7.4 Views

There are multiple windows through which client can interact with the server. Those views are as follows:

LOGIN PAGE

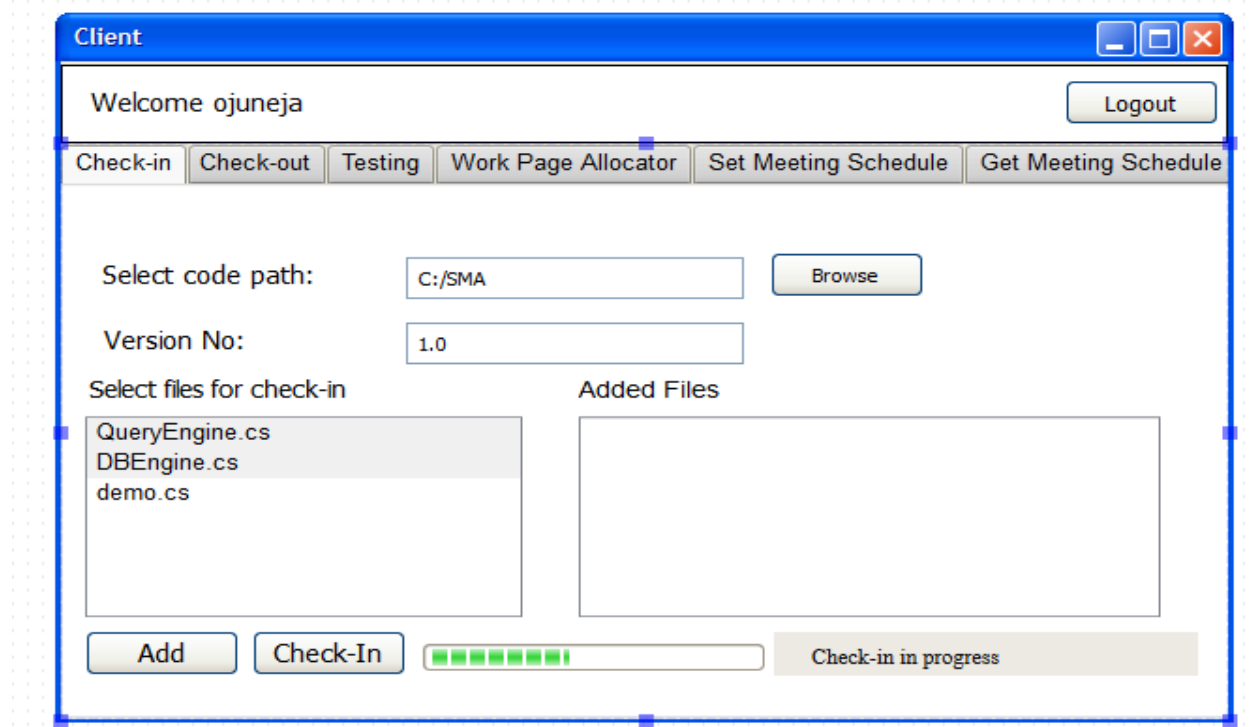


The screenshot shows a Windows-style window titled "Client" with a blue title bar and standard minimize, maximize, and close buttons. The window content has a white background and contains the following elements:

- A welcome message: "Welcome! Enter your credentials to login!"
- A "UserName" label followed by a text input field containing the text "ojuneja".
- A "Password" label followed by a password input field containing seven asterisks "*****".
- A "Login" button with a light gray gradient and rounded corners.
- A checkbox labeled "Remember me" which is currently unchecked.

Explanation

There can be many number of clients like software developer, software architect, manager, tester. All are clients and having different privileges.

Check – InExplanation

The client has to select the path from which it to do check-in. After selecting the path. All the files contained in that path will be shown to the client. First the client has to add the files if the client wants to do the fresh check in. After all the files are added, then the client can do check-in. Note that, if the user has modified the file and client has not update it but checking it in inside the system. Then, it will show error in checking in. The client can also mention the version while checking in.

Checkout

If the client wants to change the modified file. Then client must take the updated file from the server, After taking file from the server. The client can edit the file and then start the check – in process as mentioned in check-in. the checkout UI is shown below. The files present will show which files are present on server and modified files will show which files are modified. Note that the checkout happens from the server. So, its like downloading the file first from the server and then checking it back into the server.

Client

Welcome ojuneja

Logout

Check-in

Check-out

Testing

Work Page Allocator

Set Meeting Schedule

Get Meeting Schedule

Select Module

Kernel Module

Files Present

Linker.cs
d3.cs
Allocator.cs
p3.cs

Modified Files

Linker.cs
p3.cs

CheckOut

Checkout in Progres!!

Test Harness

Client

Welcome ojuneja

Logout

Check-in

Check-out

Testing

Work Page Allocator

Set Meeting Schedule

Get Meeting Schedule

Test Configuration Number

123e4

Get Configuration Numbers

Environment

qa

Get Build Numbers

Browser (Optional)

Server Response

Version

Update To

Build Number (Optional)

Test

Explanation

The client can see the configuration details, build number details either from quality center or from the GUI present above

It is the responsibility of the client to send the test config number in order to test the test cases.

The testconfig number will be mapped to the particular set of test cases. The QC will have all the structure of modules, test cases which is mapped to a particular test config number. After pressing the test button, the request will be sent and test cases will run on server and client will get the result. The response will be the report showing test cases fail and test cases pass.

Version is the version number of the tool that must be specified by the client.

Update to is the also the version number if the client wants to update the tool to any other version.

WorkFlow Package Allocator

The screenshot shows a web application window titled "Client". It has a blue header bar with standard window controls (minimize, maximize, close) on the right. Below the header, there's a "Welcome, ojuneja" message and a "Logout" button. A horizontal menu contains several tabs: "Check-in", "Check-out", "Testing", "Work Page Allocator" (which is the active tab), "Set Meeting Schedule", and "Get Meeting Schedule". Under the "Work Page Allocator" tab, there are four input fields: "Package" (a dropdown menu showing "Package1"), "Person" (a dropdown menu showing "abc"), "Task" (a text input field), and "Description" (a text input field). Below these fields are two buttons: "Retrieve Work Package Information" and "Store Work package Information". To the right of the input fields, there's a section titled "Dependent Packages" containing a list box with "Package 2" and "Package 3". Below that is a section titled "Server Response" with a large empty text area. At the bottom left, there's a progress bar with four green segments.

Explanation

Work flow package allocator will allocate the package to the person. The client can allocate the package and the task associated with that task to developer. The client can also see the work package allocator information. To store or to retrieve the workflow package allocator info. The client has to fill the package. In order to get workflow info, there is no need to fill the person or set its task and description. But to store the workflow package information. It is necessary to set the task, store the description and set the person's name.

Set Meeting Schedule

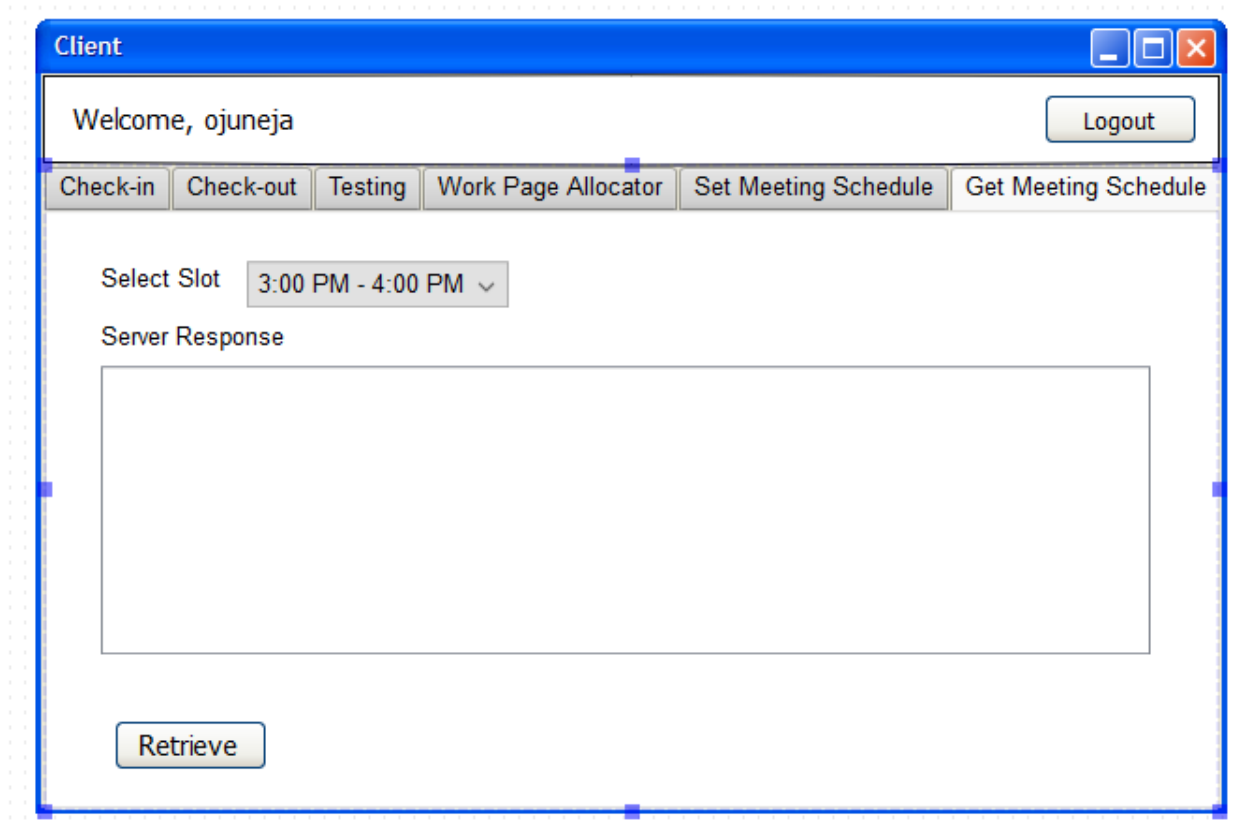
The screenshot shows a web application window titled 'Client'. At the top, there is a welcome message 'Welcome, ojuneja' and a 'Logout' button. Below this is a navigation bar with buttons for 'Check-in', 'Check-out', 'Testing', 'Work Page Allocator', 'Set Meeting Schedule' (which is highlighted), and 'Get Meeting Schedule'. The main content area is titled 'Set Meeting' and contains a 'Select Employees' button. Below this button is a list of selected employees: 'Employee1' and 'Employee2'. To the left of this list is a form with the following fields: 'Enter number of employees:' with a value of '06', three checkboxes for 'Show Code Window', 'Show Productivity Window', and 'Show Action Items' (all are unchecked), and a 'Meeting Start time and date:' field. At the bottom of the form is a 'Create' button. A status message 'Custom meeting created' is displayed at the bottom of the window.

Explanation

Client can initiate request for meeting. Number of employees with the employee name should be mentioned. It is the addition functionality that it can mention whether to show code window, productivity window and show action items or not. Meeting time and date should also be mentioned.

Get Meeting Schedule

Get meeting schedule is used to get the meeting schedules based on time slots. The output will be the meeting schedules which contains the person involved in the meeting and the things and key points of the meeting. Not every person can retrieve the meeting schedules. Only authorized people can do it. Below is the view showing the GUI.



8. Policies

The proposed principles and policies of Software Collaboration Federation are discussed below. Some of them are described in details in the following sub sections:

8.1 Asynchronous Behavior for Networking

The policies shows that all the tasks that are related to networking are handled asynchronously. Because the networking on main thread can freeze the GUI and slow down interaction. So, this approach is followed. Also, all the task by the Repository and Test harness server handled asynchronously.

8.2 Code Commit Policy

The code must be committed if the client takes the code from the updated file. The code commit policy says that no code should be committed if not updated. We follow checkin and checkout policy according to which a modified file should be checkout first before checking it in again. This will allow multiple users to commit the file but not simultaneously.

8.3 Handling Concurrent File Accesses

The locking mechanism is used for Repository to handle concurrent access on files. On the user's request of the Checked-out, Repository will lock that files. Hence, allowing only one user at a time to work on current version of a file.

8.5 File Caching Policy

To improve the performance, File Caching is the most important factor to be considered. The cache memory is used when the client or the server is sending the data of bytes from one endpoint to another. The data is transferred in chunks.

8.6 Check In policy

Only authorized users called owners are allowed to check-in new modules. Other users will just extract the components. The detailed policy regarding check-in is explained in Repository Server section.

8.7 Notifications

Notifications are the events indicators that notify the different subsystems of the SCF regarding the intermediate request or result.

User has the functionality to view the notification, mark as read, and mark as unread, notify me later, etc. The notifications will be done by the event manager of the Storage management Subsystem

The type of notifications are:

1. Notification about sending test suit pass or fails notification from the Test Harness Server to Repository Server and Client.
2. Notification about sending notification after generating the successful build image from Build Server to Test Harness Server.
3. Notification about sending notification to different team leaders for building events, meeting requests.
4. Notification about sending check-in success or failure notification to the end user from the Repository Server.
5. Notification regarding new code version release or modification to an existing code version to the collaboration server.

8.8 Ownership

Only authorized users can do specific tasks. Check-in and checkout should be done by the authorized users only. Before checking code-in, if the developers is not having the privileges to the repository. They can ask their managers to provide the privileges in order to check-in. Also, the workflow package allocator cannot be allocated by every member of the team. It is the responsibility of the team leader or team manager to do that.

9. Services

Communication Service

Communication service is very essential component in the Storage management subsystem. We will use WCF for all communications between clients and servers or server to server. Using WCF, we can send data as asynchronous messages from one service endpoint to another. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML.

Some of the features of WCF are:

Service Orientation

Interoperability

Multiple Message Patterns

Service Metadata

Data Contracts

Security

Reliable and Queued Messages

Durable Messages

Transactions

Notification Service

Notification Service is provided at both ends client and server. The notification is handled by the event handler of storage management subsystem. The event handler contains event sender and event receiver.

10. Tools

10.1 Version Control Tools for Software Collaboration Federation

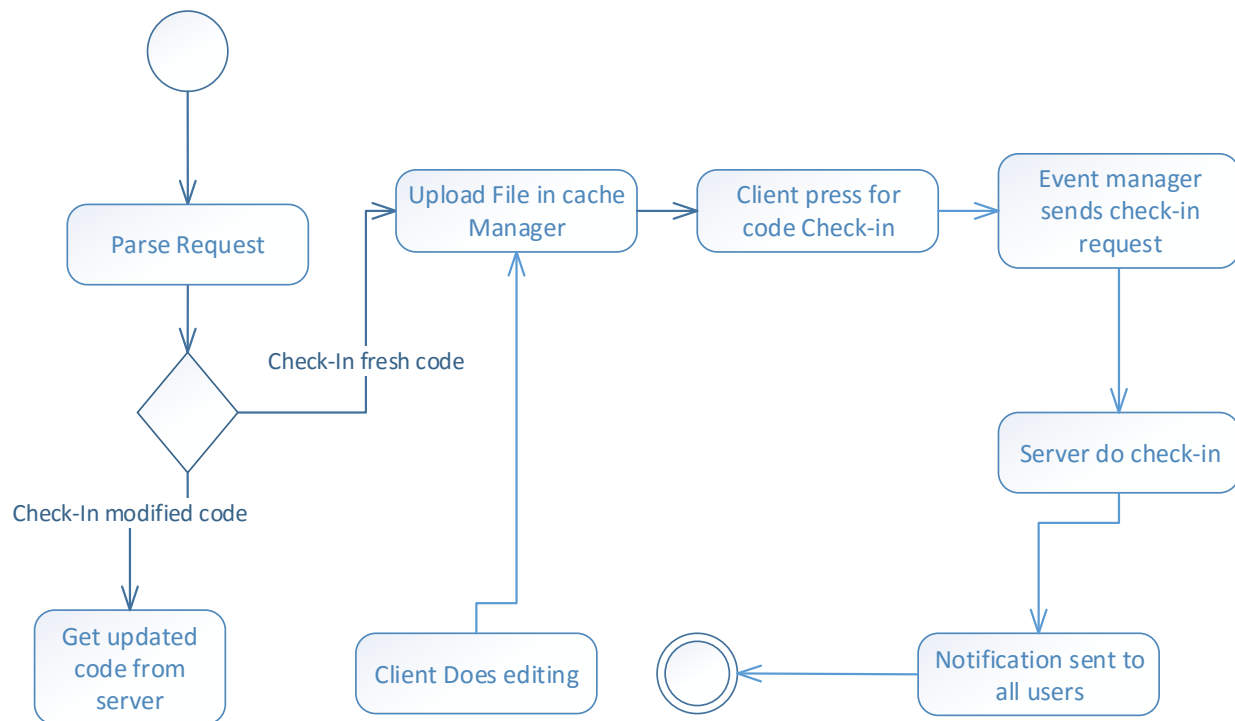
In most of the software industry, the versioning tools are wrappers that are built on top of open source tools. For example, the tools used for our SCF is a tool that has mostly similar features with tortoise SVN. I will explain the major tasks like check-in and check-out.

Check-In and Checkout

The check-in can be of two types:

1. Type1 is checking the fresh code. The packages which are required for checking in are selected and all the things are added. Here, added means the files are uploaded in the cache manager. The file is uploaded as bytes of data. Now, those data is added to the repository of the client once the client presses the check-in button. If there is any file which client is checking in is modified already by some other person. Then, the file would not be check in. A message will be shown that client copy is not updated.
2. Type 2 is check-in of code after checkout. If any other person is checked in then all the person in the same module gets the notification automatically. The person take the updated module from the repository and start editing. If both person takes the updated file, then only one person can commit. The other person can send his changes to the person which is committing and then he/she can commit.

The above two points is clearly shown in the below activity diagram.



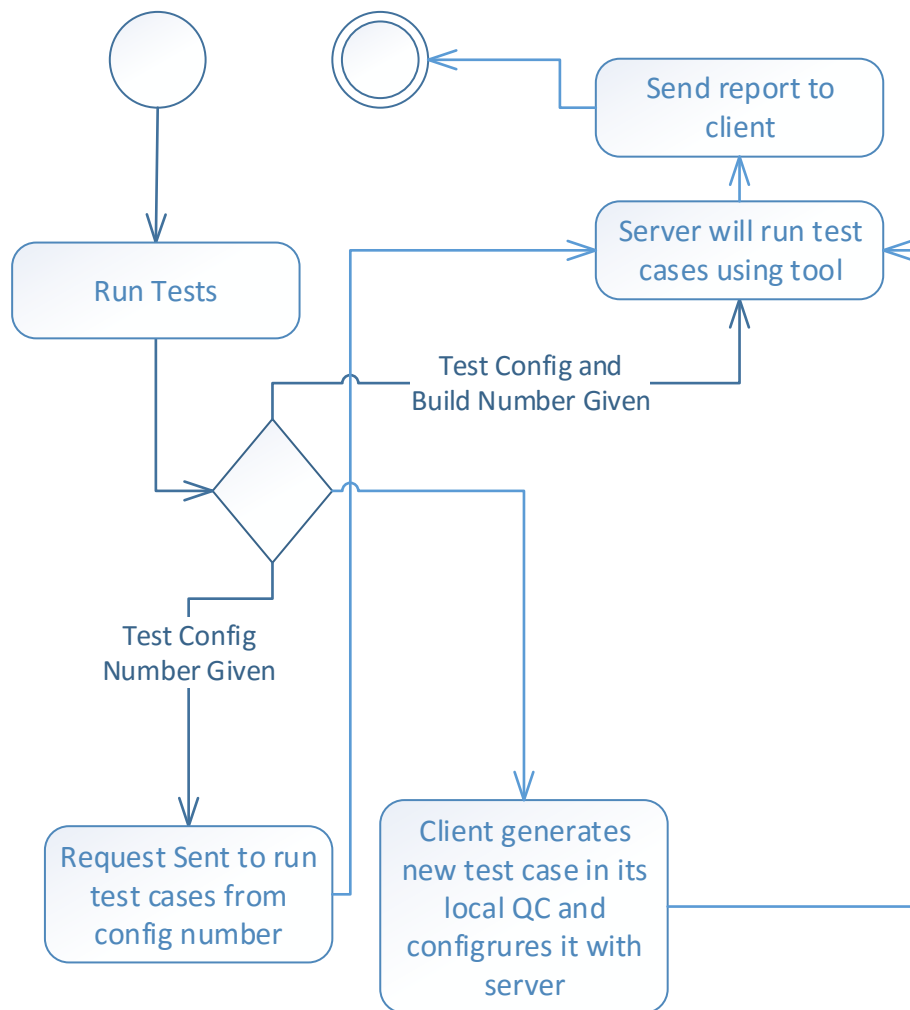
10.2 Testing Tools

The test harness server uses the built in tools to run test scenario. There are many parts of software. Some parts may require web tools for testing where as some requires functional testing and regression testing tools. Not only this, we require tools for managing the test cases. We use Quality Center for managing the test cases.

10.2.1 Workflow

The client may supply the test configuration number through which the tool can detect which test cases to run. Client can also supply old build number which he/she wants to run. The client can simply specify the test cases without mentioning build number. Client must specify either test config number or test cases, tool. If client will supply both config number and test cases then server automatically run test cases mapped to the config number if it exists. It is the server responsibility to generate config number for client and update it in test management tool. Note that, there is one critical problem associated with this procedure. The client has QC at local and the server is having QC remotely. Both needs to be synchronized. It may happens that some other tester1 change the test requirements but the tester2 is unaware about it and causes chaos in the team.

The complete workflow is already explained in below diagram. After all the test cases run, the client will be notified.



10.3 Bug Analysis Tool

The Quality Center has a defect mechanism from which manual testing team can raise the defects. Each defect will be mapped to test config number and to be very precise, each test case. If any defect is raised. Then there will be a tester assigned for a couple of bugs who will track the bugs and contact with the developer if it is not solved for a long period of time. However, if the bug is raised, the developer will also be notified. These Bug analysis tools can be used in functional and regression testing. However, there are different tools that are used to find little bugs and defects in code.

11. Conclusion

This architecture report describes how the remote key value database will be used in implementing a system. With keeping in mind the solution of the critical issues, the implementation of this system will not contains the major flaws. In conclusion, SCF provides following features:

1. Provides User Friendly GUI for interaction with the system
2. Provides enhancement and modification capabilities to the exiting project.
3. Provides option to select different functionalities such as Check-in, Check-out, Key Value Storage, View Reports, Scheduling Meetings, Add work packages, Virtual Display System.

We can summarize Software Collaboration Federation as an efficient and robust tool used for storing large amount of unstructured data.

12. References

1. <https://piazza.com/class/idvr3t7iowm3zh?cid=212>
2. <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/cse681codeL25.htm>
3. <https://piazza.com/class/idvr3t7iowm3zh?cid=196>
4. http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/BestFinalProjects/Arunkumar%20Manikantan_QAT%20OCD.pdf
5. <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/lectures/Project5-F2015.htm>
6. <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project5HelpF15/HelpForProject5.pdf>
7. <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/projects/Pr5F12.pdf>
8. https://en.wikipedia.org/wiki/Software_agent
9. <http://www.codeproject.com/Articles/166763/WCF-Streaming-Upload-Download-Files-Over-HTTP>
10. <http://pencil.evolus.vn/>