QuASE Architecture

Version 1.0

Klagenfurt 2015

Contents

1 QuASE components: an outline

QuASE architectural components are shown on Fig.1. In the following sections, these components will be described in detail.

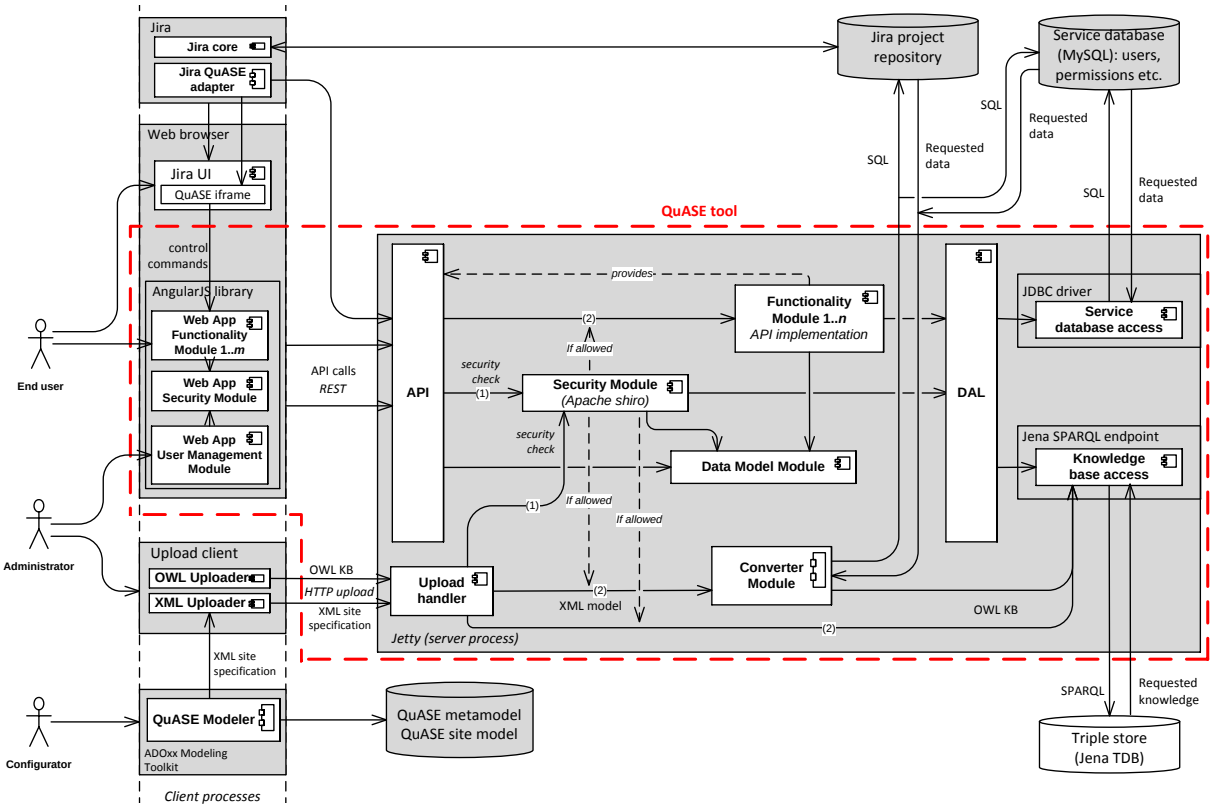

Fig.1. QuASE system architecture

2 QuASE tool components

On Fig.1, the QuASE tool boundaries are enclosed into red dashed outline. The tool contains server and client components, these components are described below.

2.1 QuASE server

Main functionality of the QuASE tool is implemented by means of *QuASE server.*

The server is based on jetty HTTP engine and can be accessed from anywhere over intranet or Internet: it listens to the dedicated TCP port. It can be accessed

1) Via REST-based API from HTTP clients;
2) Via uploading the files by means of HTTP upload mechanism, there exists the necessary infrastructure (upload handler) for processing uploaded files. Based on the upload mechanism, the tool supports an infrastructure to upload ontology OWL files and XML model files (see section “QuASE Modeler tool” below for a means of producing such files).

To implement the functionality exposed through API, the set of interfaces is available. To implement the particular subset of API calls:

1) The specific class derived from SecureFunctionality (*functionality module*) has to be defined with the methods implementing API calls; the mapping between the methods and the calls has to be specified;
2) The functionality module has to be registered in the application as a HTTP protocol handler, then the API implemented by this module could be called via particular directory URL e.g. *http://server:8085/understandability* while specifying the name of the call and the REST parameters;

The following categories for functionality modules can be distinguished:

1) Security modules;
2) The modules for context and document processing;
3) Understandability management modules;
4) Analysis modules;
5) Data collection modules;
6) Administrative modules.

The security subsystem relies on

1) A set of permissions related to functionality modules, in the database, the name of such module is associated to the specific permission, any attempt to call the methods of such module through API call if there is no permission for that will be blocked;
2) A set of user roles related to the sets of module-based permissions (i.e. the role can be granted access to several functionality modules);
3) A set of users related to the sets of roles (i.e. the particular user can hold several roles).

The security subsystem is database-backed, a MySQL database schema with the necessary structures has been implemented as a part of the solution.

The data access layer is implemented by *DAL Module*, different components are used to access QuASE knowledge base (encapsulating SPARQL queries) and to access QuASE service database (encapsulating SQL queries). It introduces higher level of abstraction to manage and query Apache Jena TDB and the internal database including the functionality for building complex SPARQL queries.

Besides providing the storage for security data, QuASE service database is used by the QuASE tool to store the data entered by the end user (decisions, values for external attributes and user-supplied units etc.)

The business logic classes are encapsulated by *Data Model Module*, the code belonging to this module accesses external storage through DAL. The data model is shared among functionality modules. It implements a set of wrappers for individuals, data, and object properties fetched from the knowledge base.

2.2 Knowledge base support

Several architectural components support QuASE knowledge base, the most important being a converter module.

A *Converter Module* builds the QuASE knowledge base from the QuASE site model, interactive data stored in the QuASE service database, and the site data available from the project repository (i.e. Jira database).

1) A converter module first converts XML model file into OWL QuASE site ontology describing the structure of the communication environment;
2) After generating QuASE site ontology, this module collects the data from the project repository (i.e. the Jira database), and the user-supplied data stored in the QuASE service database, and converts this data into the set of OWL individuals corresponding to OWL site ontology thus forming QuASE site knowledge base.

As a result, the QuASE site knowledge base possesses the *recoverability property*: i.e. the complete KB can be reconstructed at any time based on the QuASE site model, the data from the QuASE service database, and the data from the project repository.

There are two implementations of the converter module included in QuASE system:

1) The implementation as an in-process module for the QuASE server called by its file upload handler component (this is the default use case for calling converter code);
2) The implementation as a separate command line utility that can be called remotely, in this case, the resulting OWL knowledge base can be uploaded to the server.

Based on the OWL knowledge base file generated by the converter module, a specific knowledge base support module initializes Jena TDB triple store (it is possible to both generate triple store from scratch and make incremental update).

Other components supporting knowledge base are as follows:

1) A part of DAL module features SPARQL-based KB access with connection to the model and with a possibility to write ad-hoc custom SPARQL and process the resulting data;
2) An UI module implementing internal endpoint, i.e. the interactive query window allowing the user to perform custom SPARQL queries to the existing TDB store.

2.3 Interactive QuASE client

To provide access to end users, the QuASE server is accessed by the interactive *QuASE client.*

The QuASE client is implemented in JavaScript based on modular infrastructure. It relies on AngularJS framework for implementing the modules (*Web App functionality modules*) with bootstrap for UI styles, the infrastructure includes the code to incorporate both security and API access (API and Security Web App modules) and the menu system to call the modules;

The menu system is connected to the security infrastructure, so there is possible to specify the set of roles which will see the particular menu item.

The following categories of Web App functionality modules are available (these modules correspond to menu items):

1) understandability (management and assessment);
2) analysis (similarity search, prediction, recommendation and decision making support);
3) data collection (decision collecting, recommendation assessment, external values collecting)
4) administrative modules (user management module and SPARQL internal endpoint module)

The user management Web App module supports the set of predefined roles, these roles can be assigned to the users (for non-admin users, this module degrades to simple account management capabilities with changing the user's own password and account information)

To implement asynchronous display refresh, the client and server together implement the *ping functionality*, when the client repeatedly (based on timer) invokes a particular API call (ping call) and refreshes the display based on the results of this call.

3 QuASE Modeler and upload tool

3.1 QuASE modeler tool

This tool, based on ADOxx metamodeling framework, allows to create and modify QuASE site model using graphical modeling interface, and use the information from this model and the connection information specific for the deployment site, to deploy QuASE knowledge base into the QuASE server.

This tool supports two kinds of diagrams:

1) *QuASE site model diagram* which describes the structure of the communication environment and communicated knowledge. This model does not depend on site-specific information (e.g. the type or address of the project repository database, the address of the QuASE server etc.)
2) *QuASE workspace diagram* which contains information specific to the particular site and organizes this information together with the references to the appropriate site model diagrams, in a way allowing deployment.

The QuASE workspace diagram, in particular, contains the following elements:

1) The *site configuration element* which encapsulates information related to the particular deployment site, this includes the hostname and TCP port of the remote QuASE server, JDBC URLs and user-friendly names used to access QuASE service database and the project repositories etc.;
2) The *query source element* which encapsulates information specific for the RDBMS responsible for implementing the particular project repository, it contains the set of RDBMS-specific queries indexed by the name of the query referred to from the site model. For example there could be separate query sources for Oracle and MySQL implementations of the particular repository, so moving the repository from one DB server to another will not lead to changing the site model;
3) The *site specification element* which contains the references to the particular site configuration and query source elements, and to the particular site model, e.g. “the specification for the trinitec deployment (QuASE server *qqq*) to Oracle-based repository hosted at server *xxx*”. This element contains all the information necessary for performing the deployment of the QuASE knowledge base to the particular QuASE server; its UI provides the “deploy” button initiating the deployment process.

A deployment activity launched from the modeling tool entails uploading the XML file created based on the selected site specification element (containing the information from the site model, site configuration, and query source elements), into the specific QuASE server using the upload client described below.

3.2 Upload client

The upload client is a command-line tool which connects to the QuASE server, performs the necessary authentication, and transfers the particular file to that server through user upload interface. It is capable for uploading:

1) OWL knowledge base files (e.g. generated by command line converter),

2) XML site specification files (e.g. exported by the QuASE modeler tool).

The default mode of launching the upload client is from the QuASE modeler tool (by clicking the "Deploy" button on the diagram element corresponding to QuASE site specification). In this case, the model information corresponding to the current site specification is converted into XML file which is transferred by means of the tool.

The upload client can be also called from the command line, in this case, the file containing the necessary model information has to be specified as a command line parameter. This can be useful in the following situations:

1) When the predefined model is used repeatedly without further changes e.g. by QuASE installer;
2) When the QuASE modeler is not a part of the QuASE distributive package (and the model is defined by the third party prior to installation).

4 QuASE Jira integration architecture

QuASE Jira integration architecture includes the following components:

1) The extension of the QuASE tool which allows for accepting the commands from Jira, these command contain the information about the current selected instance of the context or content unit (the selection is performed by means of Jira UI by the Jira user);
2) A Jira extension support component which extends the functionality of Jira to issue the control requests to be transferred to the QuASE tool.

These components are described in more detail in the following sections.

4.1 Extending tool support for the integrated version of QuASE

This process involves introducing the following architectural extensions:

1) The web application tier of the QuASE tool is extended to accept the control requests from the (extended) Jira and transfer these commands to the application logic tier, these requests are transferred by means of HTTP protocol as parts of the request header or the URL.
2) The application logic tier is modified to allow extending the KB queries taking into account the control request data obtained from the web application tier, these extended queries would allow e.g. to limit the set of selected KB individuals available as an input into support operation to the single individual corresponding to the Jira entity instance selected with a help of Jira UI.

These extensions are implemented through introducing the embedded mode of invoking the QuASE tool web application. The Web App functionality modules can be activated in the embedded mode; the trigger for this mode is the presence of the additional parameters in the URL query string. As rendering the UI for the Web App functionality modules includes rendering the menu system (with the links referring to URLs of functionality module views), after being activated in the embedded mode, the information obtained through the query string is passed through these links to the functional modules of the system.

4.2 Jira extension support

For the Jira to be able to send control requests to the QuASE tool, it has to be extended with the *Jira QuASE integration module* component:

1) It is invoked by Jira in all extendable UI contexts (e.g. Jira views allowing UI extension by invoking the custom code) connected to Jira entities which are defined as extendable (for Jira, such contexts could be issue views, user profile views, project views etc.);
2) On invocation, it tries to find a match between a defined QuASE extension element (e.g. QuASE issue class) and an extendable Jira-specific entity class connected to the current UI context (e.g. Jira issue class);
3) If the match is found, it forms an extended Jira UI (e.g. Jira issue view) and, as a part of this process, invokes the QuASE tool by sending it a control request containing the information about the current instance of the matched context element (e.g. its unique identifier), this invocation forms QuASE UI which is then embedded into the extended Jira UI;

As a result, the user sees the *composed UI* formed by both Jira and the QuASE tool.

This component of the solution has been implemented by means of Jira plugin API as a QuASE Jira plugin. The plugin code obtains the value of the identifier attribute of the displayed instance of the matched Jira class and renders Jira UI to include the inline frame (IFRAME) HTML element. The source document of this frame is addressed through the URL of the QuASE tool web application with the URL query string containing the information necessary for turning on the embedded mode of the QuASE tool. As a result, every Jira view corresponding to the extendable element is extended with an inline frame presenting the UI of the QuASE system invoked in the embedded mode parameterized with the identifier value of the current instance of this element, the name of this element (e.g. “Issue”), and the name of its metaclass (e.g. “Content Unit”).