**Team FACEO5**

Aparup Banerjee

Hector Rosas

Laura Glendenning

Mai Nakayama

Nina Patel

|  |
| --- |
|  |
| FACEO5 Workflow Tool Software Architecture Document |
|  |
| Architectures for Software Systems, Spring 2010 |

FACEO5 Workflow Tool Software Architecture Document

Architectures for Software Systems, Spring 2010

Table of Contents

[1 Business Context 1](#_Toc260218915)

[1.1 Dynamic Network Analysis 1](#_Toc260218916)

[1.2 Project Vision 1](#_Toc260218917)

[1.3 Stakeholders 2](#_Toc260218918)

[1.4 High level use cases 3](#_Toc260218919)

[2 Architectural Drivers 4](#_Toc260218920)

[2.1 Technical Constraints 4](#_Toc260218921)

[2.2 Business Constraints 4](#_Toc260218922)

[2.3 High-Level Functional Requirements 5](#_Toc260218923)

[2.3.1 Tool 5](#_Toc260218924)

[2.3.2 Service Selection 5](#_Toc260218925)

[2.3.3 Back end 5](#_Toc260218926)

[2.4 Quality Attributes 6](#_Toc260218927)

[2.5 Analysis 6](#_Toc260218928)

[3 Architectural Description 8](#_Toc260218929)

[3.1 High-Level Architecture 8](#_Toc260218930)

[3.1.1 Dynamic View 8](#_Toc260218931)

[3.1.2 Entity Responsibility Catalog 9](#_Toc260218932)

[3.1.3 Relationship Responsibility Catalog 10](#_Toc260218933)

[3.2 Low-Level Architecture 10](#_Toc260218934)

[3.2.1 Dynamic View 10](#_Toc260218935)

[3.2.2 Element Responsibility Catalog 12](#_Toc260218936)

[3.2.3 Relationship Responsibility Catalog 15](#_Toc260218937)

[3.2.4 High Level Use Case to Sequence Diagram to Mapping 15](#_Toc260218938)

[4 Design Analysis 20](#_Toc260218939)

[4.1 Summary Table 20](#_Toc260218940)

[4.2 Impact of the 3-tier Architectural Style Constraint 21](#_Toc260218941)

[4.3 WireIt Framework 22](#_Toc260218942)

[4.4 DWR Framework 24](#_Toc260218943)

[4.4.1 Trade-Offs 25](#_Toc260218944)

[4.4.2 Analysis 26](#_Toc260218945)

[4.5 Intermediate Workflow Language 26](#_Toc260218946)

[4.6 Analysis Plug-ins as Web Services 27](#_Toc260218947)

[4.6.1 Trade-Offs 27](#_Toc260218948)

[4.6.2 Analysis 29](#_Toc260218949)

[4.7 Google Gears 29](#_Toc260218950)

[5 Appendix A: Technical Glossary XXXI](#_Toc260218951)

[5.1 Data Formats XXXI](#_Toc260218952)

[5.2 Protocols XXXI](#_Toc260218953)

[5.3 Languages XXXII](#_Toc260218954)

[5.4 Message Passing XXXII](#_Toc260218955)

[5.5 Other XXXIII](#_Toc260218956)

[6 Appendix B: Complete Utility Tree XXXVI](#_Toc260218957)

[7 Appendix D: DWR Technical Notes XXXVIII](#_Toc260218958)

[8 Appendix F: Links to Other Resources XXXIX](#_Toc260218959)

[9 References XL](#_Toc260218960)

Table of Figures

[Figure 1: Context Diagram 2](#_Toc260218961)

[Figure 2: Dynamic View of High-Level Architecture 8](#_Toc260218962)

[Figure 3: Decomposition Mapping of Dynamic View 11](#_Toc260218963)

[Figure 4: Dynamic View of Low-Level Architecture 12](#_Toc260218964)

[Figure 5: Sequence Diagram for User Login 16](#_Toc260218965)

[Figure 6: Sequence Diagram for "Get Services" 16](#_Toc260218966)

[Figure 7: Sequence Diagram for "Save Workflow" 17](#_Toc260218967)

[Figure 8: Sequence Diagram for Workflow Execution 18](#_Toc260218968)

[Figure 9: Sequence Diagram for Progress Monitor 19](#_Toc260218969)

The purpose of this report is to describe our architectural work on our Studio project. In it, we propose and document an architectural design and analyze the relative merits of that design. Our goal is to practice key principles presented in the course and foster deeper thinking about what motivates architectural design decisions, practice architectural reasoning, and architectural documentation.

The report is laid out in the order specified by the assignment description. First, we describe the business context of our project, including the domain (Dynamic Network Analysis), the project vision, and stakeholders. Next, we describe the architectural drivers of the system, including the technical and business constraints, high-level functional requirements, and quality attributes. The following section describes our current architecture, including a high-level dynamic view along with a low-level dynamic view. Lastly, we present an analysis of our design, including a description of and trade-off analysis for a number of our key architectural decisions.

# Business Context

## Dynamic Network Analysis

Dynamic Network Analysis (DNA) is a breed of information analysis that studies social structures that are extracted from natural text data. These analyses typically require a number of different techniques, including: natural language processing, extracting key relationships between entities in the social structures, and analyzing and visualizing this information as an ecology of evolving networks including social, knowledge, and activity networks. [[1](#Gar09)] For example, as described in [[1](#Gar09)], such tools can be used by scientists to understand change in the Sudan, military or intelligence agencies to understand how to interact with allies, organizational analysts to examine changing connections among firms and products as evidenced by news stories.

Here at Carnegie Mellon University, the Center for Computational Analysis of Social Organizational Systems (CASOS) is a research lab that studies dynamic network analysis. It has developed a number of tools to aid analysts who use DNA on social network data. However, these tools have a number of limitations. They require extensive training and technical expertise because of complicated and non-uniform interfaces. The set of steps used to perform an analysis, called a *workflow*, is not tracked by the tools and so are lost. Lastly, this analysis workflow cannot be shared with others, which makes collaboration difficult.

## Project Vision

To address these limitations, the CASOS lab and the Institute for Software Research (ISR) has developed a joint vision as follows:

1. Provide the CASOS tools’ capabilities in the form of composable services.
2. Host these services on a service-oriented architecture called SORASCS (Service-Oriented Architecture for Socio-Cultural Systems).
3. Develop a front end as a workflow management application that interfaces with SORASCS. This will allow non-technical analysts to efficiently perform DNA by orchestrating the services in the form of a workflow. It will allow them to create, execute, save, and reuse their analysis logic in the form of a workflow that can also be shared with others.

The FACEO5 project is to develop the workflow management application described in the third step.

## Stakeholders

There are a number of relevant stakeholders:

* Team FACEO5: the MSE team responsible for developing the workflow management application.
* The clients (Dr. David Garlan and Dr. Bradley Schmerl): project champions, customer proxy, and technical lead.
* The SORASCS development team: part of the ISR and responsible for developing SORASCS.
* The CASOS lab: responsible for the tools and interacting with the end users.
* The end users: members of academia and military who use DNA in their work. Non-technical with regard to computers, but very knowledgeable within their field.
* Project Funders: Department of Defense (DoD), a military organization.

The project vision can be represented with the context diagram shown in .

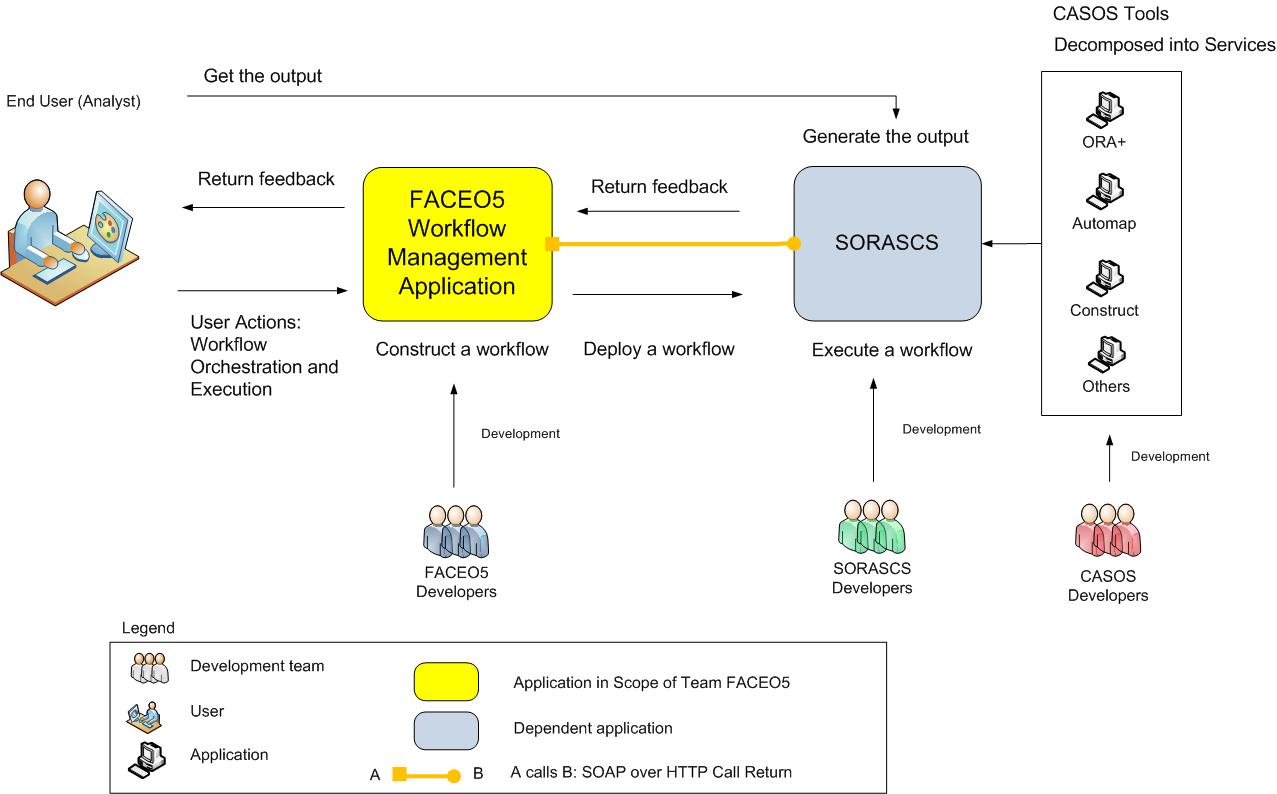


Figure 1: Context Diagram

## High level use cases

The purpose of these use cases is to give a general idea of the main activities that an analyst will perform using the workflow management application that team FACEO5 is developing (as listed in the context diagram above). These can be used later in the report to analyze how well our architecture supports these needs. For the sake of brevity, not all of our use cases are included here.

|  |  |
| --- | --- |
| Use Case Name: | **Workflow Construction** |
| Actors: | Analyst |
| Domain Objects: | User Session |
| Description: | The user will create a workflow from scratch. |
| Preconditions: | User has logged into the application. |
| Normal Flow: | 1. The user logs into the application.  2. The user receives the list of available services.  3. The user drags and drops services in the canvas.  4. The user connects the services.  5. The user sets parameters values.  6. The user defines input information for the workflow.  7. The user sets the workflow properties.  8. The user saves the workflow. |
| Postconditions: | A workflow is created, parameterized, and saved. |

|  |  |
| --- | --- |
| Use Case Name: | **Workflow Execution** |
| Actors: | Analyst, SORASCS |
| Domain Objects: | User Session, Workflow |
| Description: | The user will execute an already created and ready workflow to get the results from the analysis. |
| Preconditions: | User has logged in the application  User has a workflow ready for execution. |
| Normal Flow: | 1. The user has a workflow ready for execution.  2. The user calls the execute action.  4. The workflow is read from database and syntactically checked.  5. The workflow is deployed on SORASCS.  6. The workflow is executed on SORASCS.  7. The user receives feedback on FACEO5 tool about the execution progress provided by SORASCS.  8. The user can access the analysis output and data that was saved during the execution of the workflow. |
| Postconditions: | The results are available to the analyst. |
| Assumptions: | The tool offers in the end of the execution a link to access the information located in SORASCS |
| Notes and Issues: | How long is the result going to be kept in SORASCS? |

# 

# Architectural Drivers

From the business context, the team has derived a number of architectural drivers. These drivers will be used when describing and justifying the decisions made throughout this document. We have divided the drivers into four categories: technical constraints, business constraints, high-level functional requirements, and quality attributes. This report is not intended to capture *all* requirements for the system: here we focus only on those that had an impact on our architectural decisions.

## Technical Constraints

* **Web-based tool**. The customer has expressed a preference for a web-based platform for the tool. The minimum browser support requirements are as follows Firefox3.0, 3.6, Chrome 4.0, IE 6.0 and 7.0.
* **BPEL** [[2](#IBM07)] **as a workflow language.** Currently BPEL is the language being used by SORASCS for the specification of the workflows for execution. In this way, BPEL is a constraint because our system must produce (or convert) workflows in this format. However, the clients have indicated that the tool should not be completely and irrevocably tied to BPEL because in the future the backend workflow execution engine (currently Apache ODE [[3](#The101)]) might change and thus require a different workflow language. Thus, we need to consider not only meeting this constraint now but also the fact that it may change in the future.
* **Interfacing with SORASCS**. SORASCS will be the part of the system where the compiled workflows will be stored and execution of them will occur. Team FACEO5 will need to communicate with it to offer the intended functionalities.
  + **Communication protocol with SORASCS.** We have a technical constraint that SORASCS functionality will be available by using SOAP over HTTP.
* **Use Open Source technologies.** The tool that we are constructing needs to be open source. Also, the selected libraries and tools will need to be open source. It is also desired that the tools have some pedigree and recognition in the software community.

## Business Constraints

* **Research project.**  This is a research project exploring efficient ways of conducting DNA. This affects the scoping of the requirements on what can be done at every moment through the development of the tool within the time frame of 16 months to deliver something to keep the funders happy.
* **Parallel/Concurrent development with SORASCS.** Team FACEO5 design decisions are influenced by the ongoing development of SORASCS; the selected technology to represent a workflow in FACEO5 the tool needs to be translated into a representation that can be understood by SORASCS. SORASCS is being developed in parallel with the workflow tool. This means that SORASCS might need to change the workflow language which they support. The workflow tool should be able to handle this without impacting the user and with minimum migration complexity.
* **Project Resource/Schedule.**  The project is being developed 5 MSE students who must finish this project within 16 months.

## High-Level Functional Requirements

### Tool

* Provide a graphical user interface to enable orchestration (construction, modification, and packaging to make it available as a new service) of a workflow.
* Execution of a workflow. At a minimum the tool must support end-to-end execution of a workflow; a lower-priority, related requirement is to allow for workflow “debugging,” which would include step-by-step execution to understand the processes that are executed in the workflow.
* Syntactic checking of a workflow. There will be set of syntactic rules that the workflow tool will operate upon to check if the workflow under construction is adhered to in order to be executable. An example of a syntactic check is to check whether the input and output port types of two connected services match.
* Save a workflow. The workflows are referenced by their names and are associated with the user that created them.
* Support user authentication through SORASCS’s access control mechanisms. On SORASCS there may be data sources or services that some users do not have access to. The workflow tool does not expose these to unauthorized users.
* Users should always be able to perform workflow construction and save operations independent of whether SORASCS or the network is offline or online.
* The tool should provide extension points to increase its functionality; the extension points should be easy to learn so other developers (FACEO5) can add extensions. There are two defined extension points: for an analysis plug-in and for a palette organization plug-in. An “analysis plug-in” defines an algorithm to analyze for certain properties of a given workflow. (This is described in more detail in *Section* .) A “palette organization plug-in” defines a different GUI layout for the list of services in the palette.

### Service Selection

* The user will be able to locate services in different ways (for example: filter, search, favorites, recently used).

### Back end

* Interface with SORASCS and provide user feedback. For example, providing feedback during construction for syntactic errors, and providing feedback during execution.

## Quality Attributes

In this section we describe the quality attributes of the system using a utility tree.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quality Attribute** | **Attribute Refinement** | **Scenario** | **Stakeholder Priority** | **Perceived Difficulty** |
| Performance | Backend interaction | The tool should be able to load and save a workflow of 50 services in no more than five seconds. | M | H |
| Extensibility | Analysis Extension Framework | A SORASCS developer has a new analysis to use on workflows. Given that they have developed the algorithms for this analysis, they should be able to write and register the new workflow analysis in the tool in less than one week. | M | H |
| Scalability | Fast load and backend interaction. | An analyst launches the tool. They can see the organized palette with components that are only available for the analyst in no more than 10 seconds. | M | H |
| Usability | Easy to install | The tool should be installable with less than 3 steps, and it should be apparent to the analyst whether his or her system's requirements meet the minimum requirements. If an analyst tries to install the workflow tool on a system that is not compatible, the installer recommends and can optionally download all the prerequisites. | L | L |
|  | Non-blocking | The tool interacts with SORASCS server at various points of time. At every point of time during server interaction, UI should be responsive within less than 2 seconds. | H | H |
|  | Easy to learn | An analyst who is experienced with the current CASOS tools wants to construct a workflow from scratch using services that currently exist. He or she is able to construct and test the workflow with a small data set within two hours. | H | H |

Our complete utility tree can be found in.

## Analysis

Based on these technical constraints, the team has derived the following secondary constraint. Because we are treating SORASCS as a remote system, and the clients have a preference for a web-based tool, we have an implied constraint of **three-tier architecture**. This is because a web-based tool means that the client side will be very thin and not altering SORASCS means that we can’t put business logic there. As a result, we must have a third-tier between the client and SORASCS to support the business logic. The trade-offs associated with this decision are discussed in *Section* .

Additionally, during our analysis of the architectural driver we discovered a common underlying thread: **performance**. (In this case, we are talking about performance in terms of response time.) Although it was never explicitly stated, we have learned that our system is sensitive to performance: usability and scalability, the high-priority scenarios, are very dependent upon performance. In particular, there is tension in our system because the various decisions and trade-offs that we considered almost all have some inhibition of performance. However, many of the driving quality attribute scenarios are related to an underlying assumption of high performance. Throughout the remainder of this report, we will discuss various considerations with regard to performance.

# Architectural Description

In this section we first describe our architecture at a high-level to show the overall structure of the system. Next we describe it at a lower-level so that more details can be explained. For each level of description we include a dynamic view of the system along with an entity responsibility catalog and a relationship responsibility catalog.

## High-Level Architecture

### Dynamic View

The following image shows a dynamic view of our 3-tier architecture:

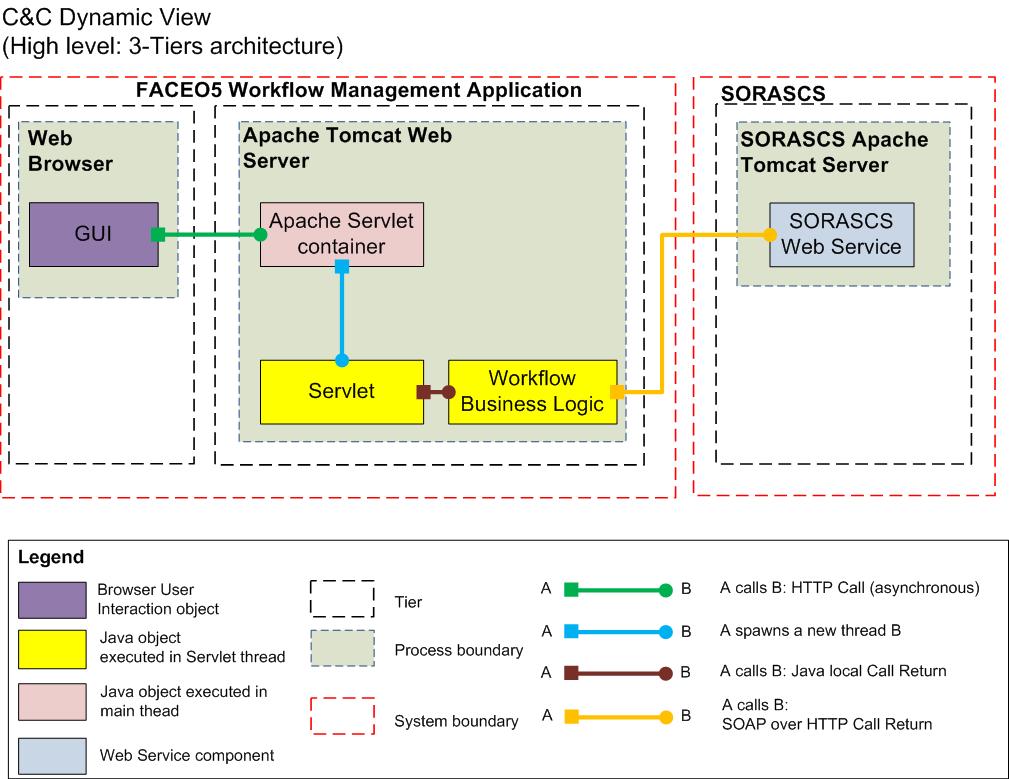


Figure 2: Dynamic View of High-Level Architecture

Decomposition rationale

|  |  |
| --- | --- |
| **Associated drawings:** **.** | **Perspective: Dynamic.** |
| As is evident from the context diagram in *Section* , the overall system will comprise of two main applications: the FACEO5 Workflow Management Tool and SORASCS. The overall system will support two main use cases: Workflow Construction and Workflow Execution (see *Section* ). Workflow construction involves graphically constructing a workflow on the “canvas” of the Workflow Management Tool; workflow execution involves executing the constructed workflow on SORASCS. Here it is important to keep in mind the preference of the clients for a web-based interface (see *Section* ). Combining these factors, from an architectural point of view we get a client-server architecture with the client on the browser communicating with the server with SORASCS. This is depicted in . Here an additional tier is introduced to house the workflow business logic, which involves workflow language generation and persistence among others. (For more detailed discussions on our choice of 3-tier style, see *Section*  and *Section* .) | |

### Entity Responsibility Catalog

This catalog is associated with drawing , which uses the dynamic perspective.

|  |  |  |
| --- | --- | --- |
| **Element** | **Responsibilities** | **Part of Workflow Tool?** |
| GUI | This element is the user interaction object executed as part of Browser process. It acts as the main user interface for the tool and provides the graphical drawing capabilities to the user. It is responsible for rendering the UI and acting as client in communicating with the web server. It is a single page interface, and is loaded only once and during web browser refresh. It is a composition of JavaScript and HTML files. This component is executed as part of the main browser process. | Yes |
| Apache Servlet container | This element is the servlet container provided by Apache Tomcat Web Server [[4](#The10)]. The servlet container is the official “Reference Implementation” for the Java Servlet. It receives HTTP requests from the GUI and then delegates them to the servlet. It is also responsible for instantiating and managing servlet threads. This component runs as a part of the main Web server process. | Yes |
| Servlet | This element receives the HTTP requests from the web server and sends the appropriate response. It is responsible for delegating the appropriate request to the workflow business logic component and then sending the response back to the servlet container. | Yes |
| Workflow Business Logic | This element is a Java object responsible for providing workflow business logic. Some examples are Workflow Generation and Workflow Execution Handling. (See *Section*  for a low-level decomposition of this.) This component is executed as part of the servlet thread. | Yes |
| SORASCS Web Service | This element is a web service provided by SORASCS. Some examples are the Web Service Registry and Workflow Execution. (See *Section*  for a low level decomposition of this.) SORASCS Web services run on another Apache Tomcat Web server (different from the server business logic components are deployed). | No |

### Relationship Responsibility Catalog

This catalog is associated with drawing , which uses the dynamic perspective.

|  |  |
| --- | --- |
| **Relationship** | **Responsibility** |
|  | The responsibility of this connector is to connect the GUI to the Web Server. This is in form of HTTP Get or Post requests from the GUI to the Web Server. |
|  | This connector is responsible for connecting two Java components. It follows call-return semantics and it is synchronous. |
|  | This connector is responsible for instantiating a thread inside a process. |
|  | This connector is responsible for connecting the Workflow Business Logic component to SORASCS Web services. It is in form of SOAP over HTTP protocol. (See *Section* 2 for the technical constraint which resulted in this constraint.) |

## Low-Level Architecture

### Dynamic View

#### 3 Tiers – High Level to Low Level Decomposition Mapping

The following image shows how the architecture is decomposed from high level to low level elements within each tier.

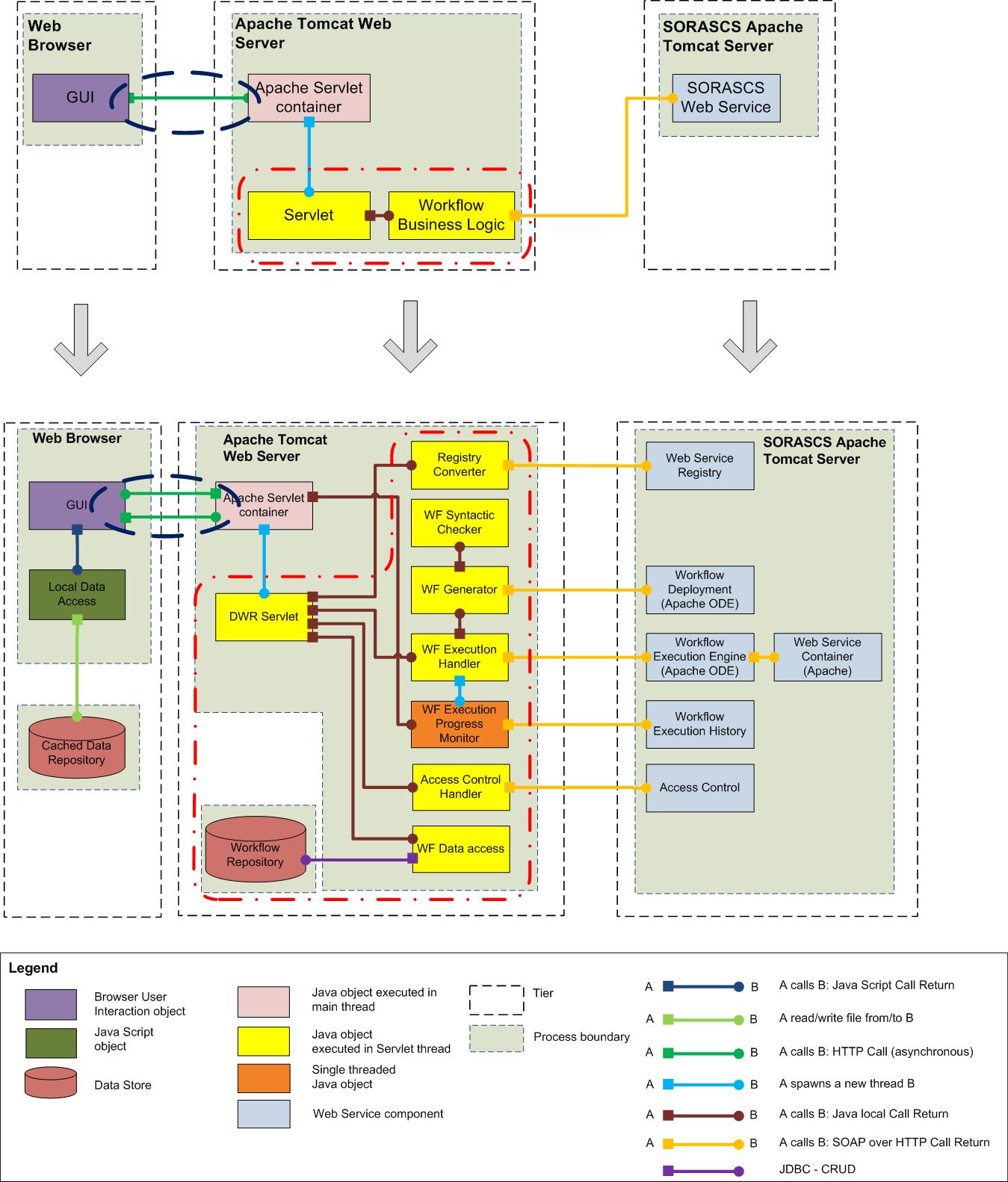


Figure 3: Decomposition Mapping of Dynamic View

Decomposition rationale

|  |  |
| --- | --- |
| **Associated drawings: .** | **Perspective: Dynamic.** |
| This decomposition focuses on the Servlet and Workflow Business logic components. The Workflow Business logic component is now decomposed into separate sub-components based on their functionality. Each of them is still a Java object. The Servlet in the high level diagram in *Section*  is now replaced with a DWR Servlet. This also replaces the connector between GUI and Apache Servlet container. (See *Section*  for more details on the DWR decision.) The SORASCS components are also broken down into sub-components to show their interaction with the sub-components of the Workflow Business Logic component. | |

#### Low Level Decomposition

The following image shows a dynamic view of our decomposed 3-tier architecture:

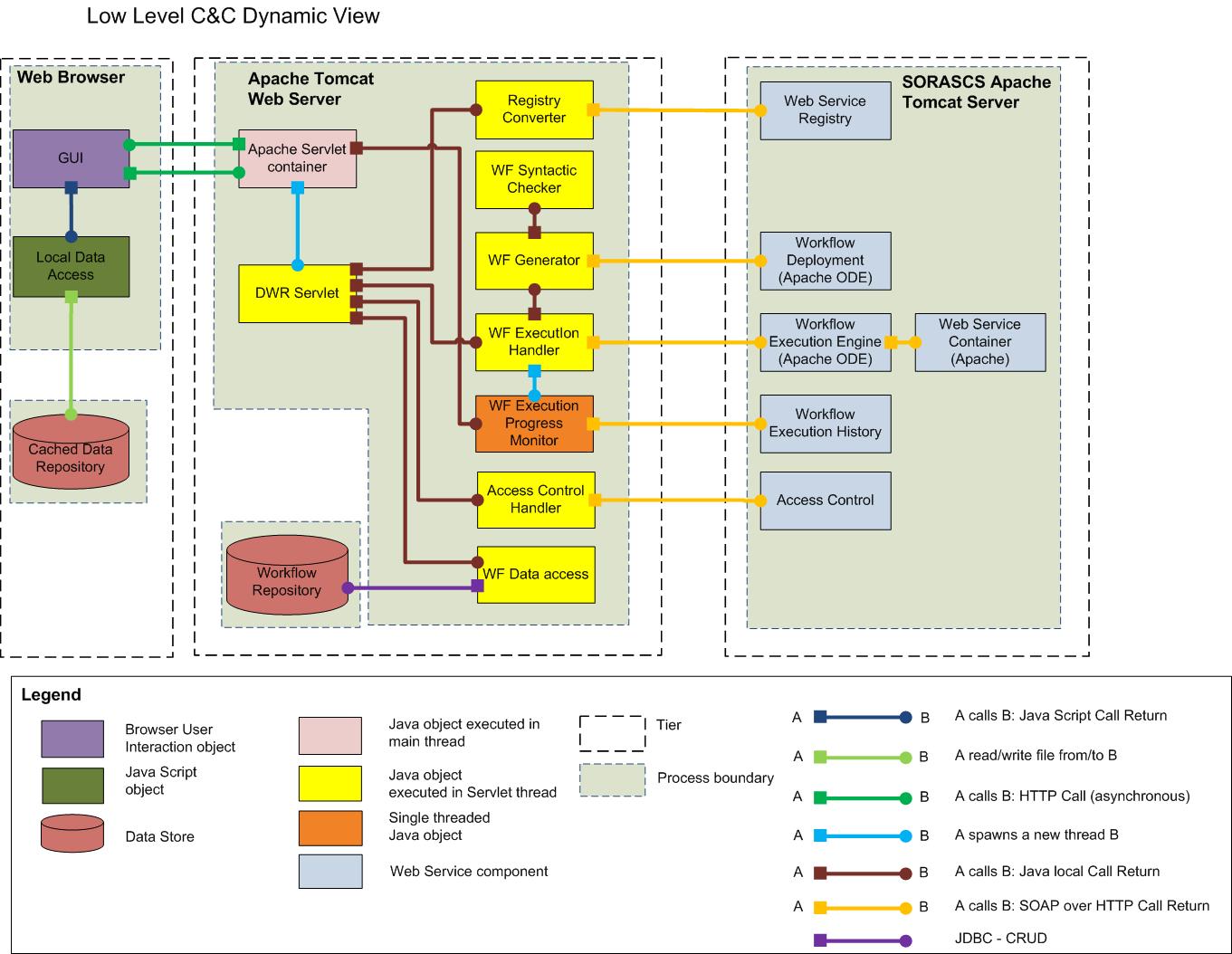


Figure 4: Dynamic View of Low-Level Architecture

### Element Responsibility Catalog

This catalog is associated with drawing , which uses the dynamic perspective.

|  |  |  |
| --- | --- | --- |
| **Element** | **Responsibilities** | **Part of Workflow Tool?** |
| GUI | This element is the user interaction object executed as part of Browser process. It acts as the main user interface for the tool and provides the graphical drawing capabilities to the user. It is responsible for rendering the UI and acting as client in communicating with the web server. It is a single page interface, and is loaded only once and during web browser refresh. It is a composition of JavaScript and HTML files. This component is executed as part of the main browser process. | Yes |
| Local Data Access | This element is a data access component. It is JavaScript object provided by Google Gears [[5](#Goo08)] to perform CRUD (create, read, update, delete) operations on the data store. This element provides a way to store and load workflow or services data to the cached data repository. (See *Section*  for local data storage decision.) | Yes |
| Cached Data Repository | This element is a data store component provided by SQLite. [[6](#SQL)] SQLite is a flat file based data store. This component is responsible for storing the cached data. | Yes |
| Apache Servlet container | This element is the servlet container provided by Apache Tomcat Web Server. It receives HTTP requests from the GUI and then delegates them to the servlet. It is also responsible for instantiating and managing servlet threads. This component runs as a part of the main Web server process. | Yes |
| DWR Servlet | This element is a servlet provided by DWR. It receives the HTTP requests from the Servlet container and then calls the appropriate workflow business logic component method. This component is multi-threaded and stateless and it is managed by the servlet container. It also sends the response back to the servlet container. | Yes |
| Registry Converter | This element is a workflow business logic component. It is a Java object. It receives the list of web services from the web service registry on SORASCS, converts the information to JSON format, and then returns the JSON back. It connects to the Web Service Registry using a SOAP over HTTP connector. This element is a stateless element and is executed as part of the DWR Servlet thread. | Yes |
| WF Execution Handler | This element is a workflow business logic component. It is a Java object. Once called, it first calls the workflow generator to generate workflow language from the workflow model. The Workflow Generator, in addition to generating the workflow language, also deploys the workflow to SORASCS. After a valid workflow package is generated and deployed, this element also calls the Workflow Execution Engine on SORASCS to start the execution. To track the progress of a workflow this element instantiates the workflow progress monitor thread. This element is a stateless element and is executed as part of the DWR servlet thread. | Yes |
| WF Data access | This element is a data access component. It is a Java object. It provides a mechanism to perform CRUD operations on the Workflow Repository through JDBC. This element is a stateless element and is executed as part of the DWR servlet thread. | Yes |
| WF Execution progress monitor | This element is a workflow business logic component. It is a Java object and is single threaded. It polls the workflow execution history web service on SORASCS to retrieve the workflow progress. It then uses reverse AJAX mechanism provided by DWR (see *Section* ) to communicate the progress back to the GUI. | Yes |
| WF Generator | This element is a workflow business logic component. It is a Java object. It is stateless and runs as part of the servlet thread. It is responsible for generating the workflow package from the workflow model and then deploying the workflow to SORASCS. It calls the Workflow Syntactic checker to validate a workflow. (See *Section*  to see the discussion on intermediate language generation.) | Yes |
| WF Syntactic Checker | This element is a workflow business logic component. It is a Java object. It is stateless and runs as part of the servlet thread. It is responsible for doing syntactic validation on the workflow. | Yes |
| Web service Registry | This element is a web service component running on the SORASCS Apache web server. It provides a read-only view of the web services registered in SORASCS. It provides information such service types, parameters, and ports from SORASCS. (Ports are the input and output data for a service; parameters are “service properties settings” to configure that service’s behavior.) | No |
| Workflow deployment | This element is a web service component running on SORASCS. It is part of the Apache ODE [[3](#The101)] workflow engine. It is responsible for deploying the workflow to the workflow engine. During this process it extracts the workflow package, and then converts into another web service. | No |
| Workflow Execution engine | This element is a web service component running on SORASCS. It is part of the Apache ODE workflow engine. Once called, it starts the workflow execution. | No |
| Workflow Execution History | This element is a web service component running on SORASCS. It provides a read-only view of the execution history of a workflow. It provides traces of service executions in a workflow along with their execution times. | No |
| Web Service Container | This element is a web service component running on SORASCS. It is provided by Apache. Once called, it executes a web service. This happens when the workflow engine executes a CASOS tools web service. | No |
| Access Control | This element is a web service component running on SORASCS. It provides the user access validation mechanism. | No |
| Access Control Handler | This element is a workflow business logic component. It is a Java object. It is stateless and is executed as part of main servlet thread. It is responsible for delegating the access validation requests to the access control web service on SORASCS and returning the validation result back to the UI. | Yes |
| Workflow Repository | This element provides RDMS-based (relational database management system) storage to the workflow. It is a MySQL database, and can be accessed using JDBC [[7](#Ora10)]. | Yes |

### Relationship Responsibility Catalog

This catalog is associated with drawing , which uses the dynamic perspective.

|  |  |
| --- | --- |
| **Relationship** | **Responsibility** |
|  | The responsibility of this connector is to connect the GUI to the Web Server. This is in the form of a HTTP Post request from the GUI to the Web Server. This connector is asynchronous. |
|  | This connector is responsible for connecting two Java components. It follows call-return semantics and it is synchronous. |
|  | This connector is responsible for instantiating a thread inside a process. |
|  | This connector is responsible for connecting the workflow business logic component to SORASCS Web services. It is in form of SOAP Over HTTP protocol. (See *Section*  for the technical constraint which resulted in this constraint.) |
|  | This connector is responsible for connecting to the SQLite database and writing and reading the data. |
|  | This connector is responsible for connecting to the MySQL database and storing and retrieving data. It is implemented using JDBC. |

### High Level Use Case to Sequence Diagram to Mapping

Below are the sequence diagrams that show the “flow” for *user login, workflow get services, save workflow* and *workflow execution*, *workflow progress monitoring* using the objects described in the *Section* ). This maps to the “high level” use cases described in *Section*  describing the project context.

#### Workflow Construction – Sequence Diagrams

The three sequence diagrams listed in this section map to the high level use of “Workflow Construction” described in *Section* .

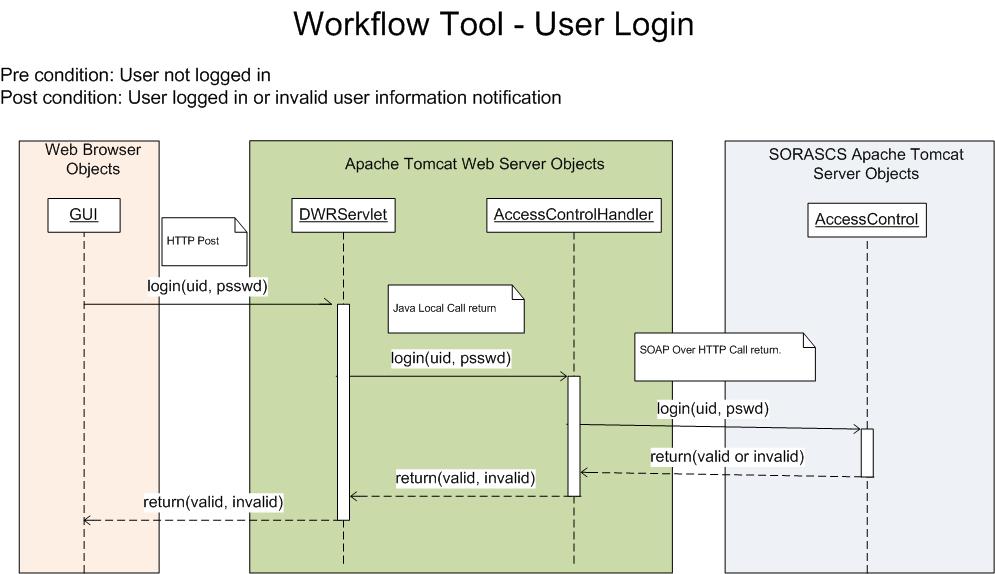


Figure 5: Sequence Diagram for User Login

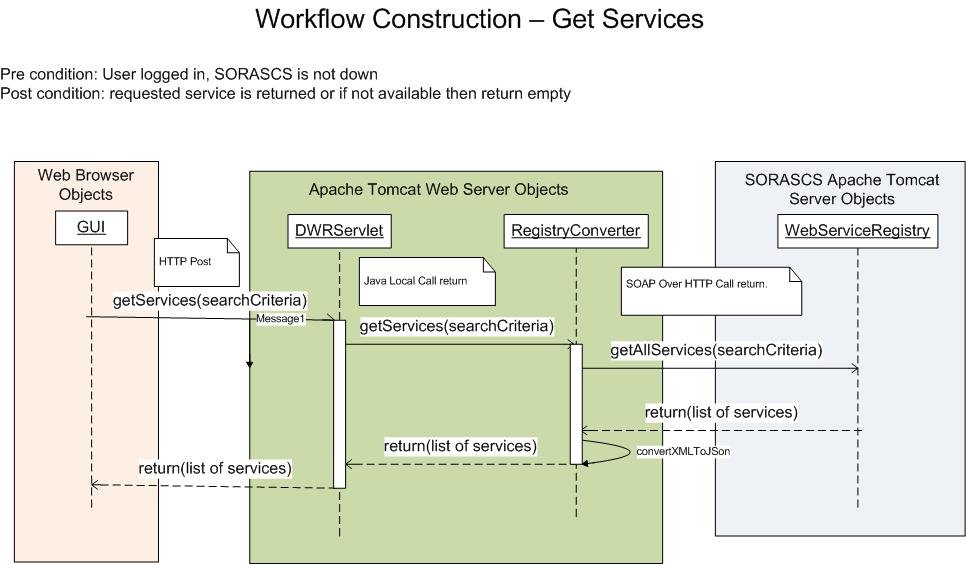


Figure 6: Sequence Diagram for "Get Services"

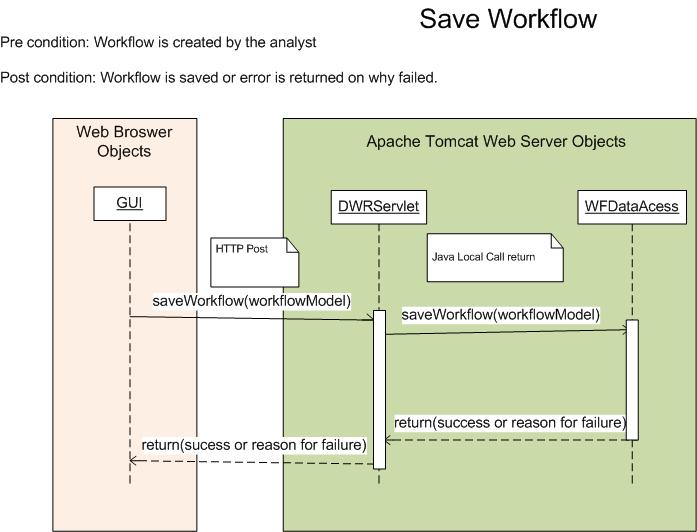


Figure 7: Sequence Diagram for "Save Workflow"

#### Workflow Execution – Sequence Diagrams

The two sequence diagrams listed in this section maps to the high level use of “Workflow Execution” described in *Section* .

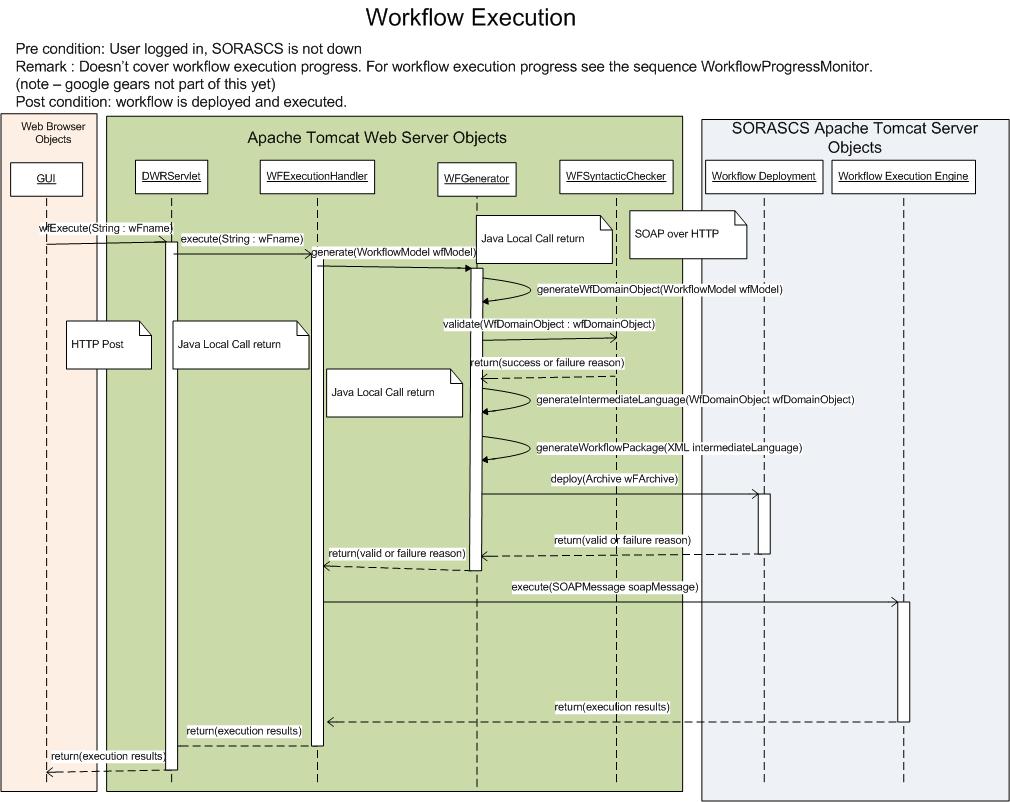


Figure 8: Sequence Diagram for Workflow Execution

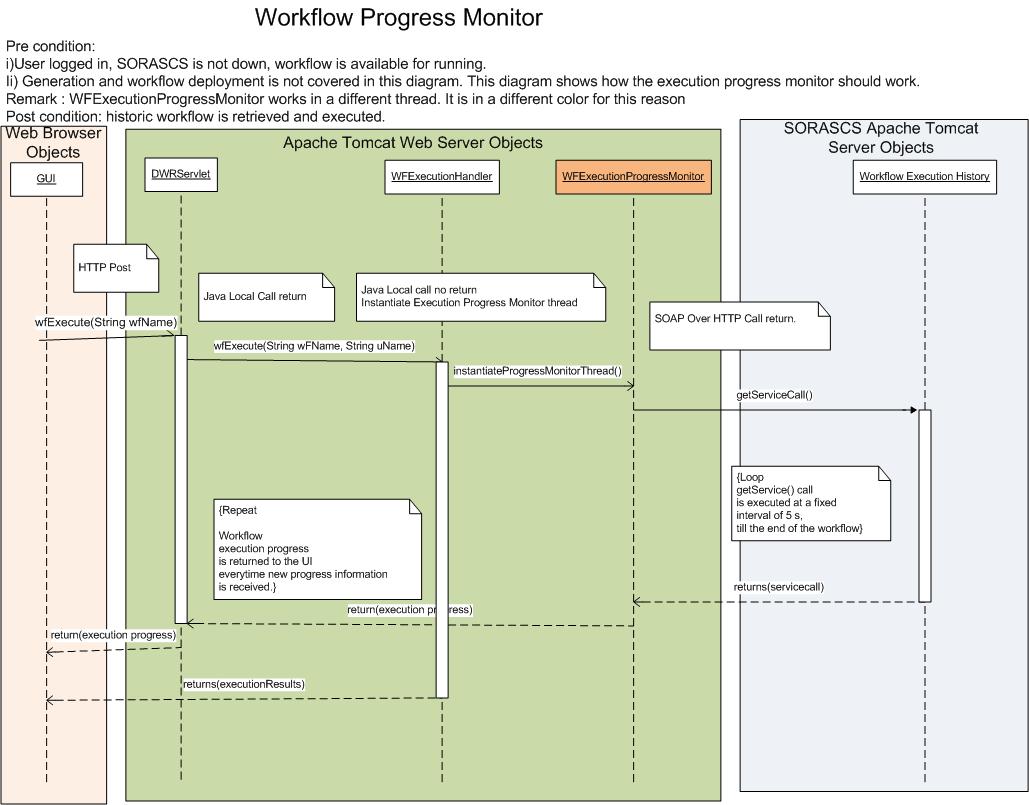


Figure 9: Sequence Diagram for Progress Monitor

# Design Analysis

Next we discuss some key decisions that the team has made with regard to this architecture. For the sake of brevity for this report, we will focus on a limited number of decisions for in-depth analysis. In this section, we first present a summary table that maps each of the high-priority architectural drivers (as described in S*ection 2* ) to the key decisions that we have made. The architectural analysis and trade-offs for each of these decisions, and any alternatives that we considered, will be discussed later in this section.

## Summary Table

Here we present a summary table, whose purpose is two-fold. First, it is meant to summarize the key architectural decisions that we have made so far for the project. Second, it is meant to demonstrate how well the project’s architectural drivers are addressed by our current architecture.

|  |  |
| --- | --- |
| **Architectural Drivers**  **(See *Section 2* for details.)** | **Key Decision** |
|  |  |
| **Functional Requirement:** Provide a graphical user interface to enable orchestration (construction, modification, and packaging to make it available as a new service) of a workflow.  **QA – Usability:** Easy to Learn  **QA – Usability:** Easy to Install  **Technical Constraint**: Preference for web-based technologies.  **Technical Constraint:** Use open source technologies. | **WireIt Framework** [[8](#Wir10)]  (See *Section* .)  Web-based graphical framework chosen after evaluation and experimentation. |
|  |  |
| **Functional Requirement:** Interface with SORASCS and provide user feedback.  **Technical Constraint:** Use open source technologies. **Business Constraint:** Project resource/schedule.  **QA – Usability:** UI responsiveness andnot block the user actions. | **DWR Framework**  (See *Section* .)  Decided to use AJAXand Reverse AJAXtechnology. Direct Web Remoting (DWR)is our choice for the AJAX Framework. See for more information about AJAX and Reverse AJAX; see  *Appendix D: DWR Technical* Notes for more information about DWR. |
|  |  |
| **Business Constraint:** Parallel/Concurrent development with SORASCS.  **Technical Constraint:** BPEL [[2](#IBM07)] as a workflow language. | **Intermediate Workflow Language**  (See *Section* .) |
|  |  |
| **QA –** **Extensibility:** Provide extension points to add analysis plug-in to check properties of a workflow.  **Business Constraint:** Project Resource/Schedule.  **Technical Constraint**: Preference for web-based technologies. | **Analysis Plug-ins as Web Services**  (See *Section* .) |
|  |  |
| **QA – Scalability:** Service load time. | **Google Gears**  (See *Section* .) |

## Impact of the 3-tier Architectural Style Constraint

Before we describe any of the key decisions, trade-off analysis based on alternatives and rationale in detail, this section briefly describes the “impacts” that the 3-tier architectural style has on our system quality attributes. In this style the workflow business logic resides on the middle tier.

**Tradeoff Analysis:**

* [**Good**] **Promotes Extensibility**

Business logic is not centralized and so in the future other tools can plug-in easily.

* [**Good**] **Promotes Maintainability**

Use of other languages for business logic (e.g. Java) is possible.

* **[Bad]** **Inhibits Performance**

There is an extra tier added. Hence, the round trip time when communicating with SORASCS increases. Thus, a separate tactic is needed to counteract this impact; this tactic is discussed in upcoming sections.

* **[Bad] Increases complexity**

Like other 3-tier architectures, the addition of a new tier necessarily increases the complexity of the system.

As mentioned in *Section* **, “performance” is the key sensitivity point** in our system and 3-tier architectural constraint doesn’t help any further. The team made a list of low level design decisions that has to be taken as tactic to counteract the bad points of the 3-tier architecture as well as address the high risk, and high priority architectural drivers. These low level design decisions are described in upcoming sections.

## WireIt Framework

Given the technical constraints mentioned in *Section* , web-browser based graphical framework technical constraint along with the **“usability” quality attribute** are the architectural drivers to choosing the “right” graphical framework. However, as mentioned in *Section* , web-browser based graphical framework effectively constrains the software architecture to the 3-tier architecture style. This has implications in terms of **inhibiting performance** which is the **key sensitivity point** as also discussed in *Section* .

Given this, the team decided to evaluate open source graphical frameworks for the web to see if we could find one that was suitable for our project. The client also agreed to this decision.

Also, with a browser being the user interface client, we are now constrained with our choice of language on the browser. Most browsers support JavaScript as the main scripting language. A wide use of **JavaScript** can result in a lot of **“spaghetti code”** on the client side which **impacts its maintainability**. The client also expressed concern about this potential problem. However, the 3-tier architecture helps mitigate this as it will allow us to keep the client browser light-weight.

We explored the web-based graphical framework domain for potential candidates for “off the shelf” use for within our project given the “open source technologies” as our technical constraint as mentioned in *Section* . We evaluated two open source frameworks: Draw2D [[9](#And10)] and WireIt [[8](#Wir10)]. Here we will list the main results of our evaluation. The team also took inspiration from Yahoo Pipes, a web based modeling application.[[10](#Yah101)]

|  |  |
| --- | --- |
| WireIt | Draw2D |
| **[Good] Promotes extensibility**  WireIt allows for plugging in new UI elements without changing the overall structure of the framework.  This supports the “research project” business constraint given that we are still exploring new ways to conduct DNA and thus the representation of the workflow could change (for example, representing the elements as icons).  **[Good]** **Promotes configurability**  WireIt allows for configuring and customizing the user interface declaratively through JSON [[11](#JSO10)]. (For example, the service palette can be configured by constructing a JSON for it.)  **[Good] Promotes Usability - Easy user interface**  WireIt is inspired by Yahoo Pipes. (This is significant because the client had emphasized Yahoo Pipes as an example of an accepted, widely-used tool similar in its graphical environment to our tools.) WireIt provides an easy user interface for graphical modeling. The interface also allows further customization based on need.  **[Bad] Usability features**  Doesn’t support some UI features like undo/redo, zoom in/zoom out etc. Some of these items are to be addressed in future versions of the software, but it’s not clear if it will be released soon enough to help us. | **[Good] Usability Features**  Supports most of the UI features – Undo/redo, zoom in /zoom out, etc.  **[Bad] Doesn’t promote extensibility**  Draw2D is not as extensible as WireIt. The developer needs knowledge of the full API to develop a modeling environment on top of it. This API is lower-level than WireIt’s and is thus less convenient to develop with (and less extensible). |
| **In general the team found the following about both the frameworks:**  **[Good] Low installation overhead**  The tools are targeted for non computer science specialists, who don’t want to perform complicated installation and configuration tasks. If browser is the only medium of interaction, then there is no installation overhead for the end user.  **[Good] Greater platform independence**  The tool being web based also supports working from variety of platforms like mobile devices, internet kiosks etc.  **[Bad] Challenging to provide modeling environment on web**  The team found that web browsers can be a challenging platform to provide a modeling (partly drawing) environment. The widgets and the user interaction capabilities on the browser can be limited and may not be as rich as what is available on the desktop environment.  **JavaScript usage** – Both the frameworks are developed using JavaScript as the scripting language. While JavaScript itself is a powerful language, it can lead into maintainability issues because it doesn’t inherently support modularity in the same was as object-oriented languages like Java.  **Community support** – Both WireIt and Draw2D are evolving frameworks. Community support for both of them are not at the same level as matured frameworks like Eclipse or Apache. | |

#### Analysis

**Decision: WireIt Framework.**

The team conducted experiments with WireIt. Those experiments were based on the following aspects:

1. Get to know the capabilities of the user interface of WireIt.
2. Test the extensibility of WireIt.
3. Explore ways to interact with the server-side of the application using WireIt (which gave rise to the low level design decisions mentioned in the upcoming sections).

Based on the results of those experiments, the team decided to go with WireIt as our web-based framework. However, the team also expects an implementation effort for customizing WireIt as needed because we are using a third-party software source. We must keep in mind that we may have to change the library’s code ourselves, or communicate with the WireIt developers to do so.

## DWR Framework

As we have discussed previously, our system is sensitive to performance (and usability). We have already discussed one major area in which performance is a key concern: the three-tier architectural style. In this section, we will discuss two other areas: synchronous vs. asynchronous communication and message passing techniques. We looked for open-source technologies that we could use in our project to address some of the potential issues in these two areas.

As listed in *Section* , usability, resource/schedule constraints, open-source technology constraints, and the feed-back loop with SORASCS are the architectural drivers driving the decision of DWR. Also, as discussed in *Section* , it was decided to have a 3-tier architecture for the system. As mentioned in the tradeoff analysis of that section, this inhibits the overall system performance by increasing the number of required network round-trips: it has increased from 1 to 2 (UI 🡪 Middle tier 🡪 SORASCS). Therefore we need a tactic to counteract this problem. The decision of DWR is motivated by this high-level design choice along with the architectural drivers mentioned above.

First, we decided on the communication strategy with the middle-tier. Would it be asynchronous or synchronous? This is especially important because we have a web-based framework and usability as a driver. To illustrate this point, consider a typical example of a user interacting with our tool: when an analyst chooses to execute a workflow. A DNA workflow can take hours to complete. Given that, for a full execution, synchronous communication will end up blocking the user operations which could result in user dissatisfaction. Hence, asynchronous communication is better choice in this case because it promotes usability (as the user is not blocked while a long network operation is underway).

However, asynchronous communication reduces predictability while communicating with a backend system because now the client must listen for response events sent by the server (also known as “callback”). It could be difficult to trace which server response is from which request by the client. However, there are effective ways of handling this on web applications with AJAX frameworks: for example, some frameworks associate a unique id with each request so that there is traceability between request and response.

Second, we decided on the best strategy for message passing techniques to convey the “workflow execution progress” to the user. Our choice was to use “reverse AJAX” techniques for communicating between a web server and browser. **(**Reverse AJAX refers to an Ajax design pattern that uses long-lived HTTP connections to enable low-latency communication between a web server and a browser.) Current AJAX frameworks implement Reverse AJAX in using the following message passing techniques:

* **Client polling:** The client repeatedly queries (polls) the server and waits for an answer.
* **Server push:** A connection between a server and client is kept open and the server sends data when available.
* **Piggyback:** Piggyback is a comparatively new model used in some AJAX frameworks. The piggyback approach, as the name suggests, is the “piggy backing” of new data on top of an unrelated response to a client’s request. When new information arrives at the server, it is stored until the client next makes a request to the server.

Each of the above techniques had pros and cons with respect to performance (latency), complexity, and server scalability (overloading the server with concurrent requests). Given all the data above, we found that Reverse AJAX is an effective way to communicate from server to the client to satisfy the architectural drivers. The optimal choice would be to be able to configure the message passing mode by the AJAX/Reverse AJAX framework which would be suitable to **promote performance and usability concerns**. With all these factors in mind, the team was looking for an AJAX framework which is easy to use and is configurable in terms of Reverse AJAX options.

### Trade-Offs

We compared the DWR AJAX Framework [[12](#Dir10)] and the Google Web Toolkit (GWT) AJAX Framework [[13](#Goo101)]. Below are the main results of our trade-off analysis:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **DWR** | **GWT** |
| Usability | **[Good] Promotes Usability.**  The user is not blocked while a network operation is underway. | **[Good] Promotes Usability.**  The user is not blocked while a network operation is underway. |
| Maintainability | **[Good] Promotes maintainability.**  Allows for developing in Java and provides an easy way to dynamically convert the Java code to JavaScript code running on the browser. It encapsulates the marshalling and un-marshalling of JavaScript to Java types, and provides an easy interface to the developers. This reduces the “spaghetti” JavaScript code on the client-side, which can become a maintenance headache. | **[Bad]** **Inhibits Maintainability.**  GWT allows you to write the code in Java and then does the static generation of JavaScript and stores that code on the browser side. However, this capability is not available dynamically like DWR. It is thus hard to maintain all the JavaScript code generated by GWT. |
| Reverse AJAX | **[Good] Provides Reverse AJAX capability**  Allows various asynchronous message passing techniques, such as client polling, server push, and piggyback. | **[Bad] No Reverse AJAX Implementation**  GWT doesn’t support Reverse AJAX directly: it provides an API to the developer to implement Reverse AJAX. This increases development effort and complexity. |
| Configurability | **[Good] Promotes Configurability**  Allows you to choose the Reverse AJAX method with easy configuration options. | **[Bad]** Given that there is no “direct” Reverse AJAX support, configurability is not applicable here: it would be up to the developer to implement the configurable message passing reverse AJAX modes. |

### Analysis

**Decision: DWR Framework.**

Given the comparison data above, the team decided to use DWR. We also found some useful tools like Firebug [[14](#Moz10)] which can help in debugging web applications. This is especially useful for DWR because Firebug also allows you to debug dynamically-generated JavaScript code.

## Intermediate Workflow Language

As stated in *Section 2* , parallel/concurrent development with SORASCS is one of the business constraints on this project. Additionally, the software should generate artifacts from a workflow in a language which can be executed by SORASCS. The generated artifacts will be packaged in an archive and then deployed to SORASCS. The workflow language which is currently supported by SORASCS is BPEL; however, this may change in the future. Given this potential change, there are two approaches to generate the workflow language which SORASCS can understand: 1) directly generate a workflow language that SORASCS understands (currently BPEL), or 2) generate an “intermediate” language and let SORASCS be responsible for having the necessary converters from this intermediate language to their “chosen” workflow language.

The table below lists thetradeoff analysis between having intermediate language vs. doing direct BPEL generation. There is also a schematic representation of the language generation process after the table.

Question: **Which language generation approach to use?**

|  |  |
| --- | --- |
| **Intermediate Workflow Language** | **No Intermediate Workflow Language** |
| In this approach, the team and client must agree on an intermediate language which the tool should always generate. The tool must then provide plug-in points for generation of additional languages from the intermediate language. Thus there are two generators in this approach: one which generates an intermediate language from JSON, and a second convertor which converts the intermediate language to a language that the SORASCS execution engine understands. (Currently, SORASCS’s workflow language is BPEL.)  **[Good] Promotes extensibility (given the business constraint of SORASCS parallel development)**  The introduction of the intermediate language allows the tool to plug-in multiple language generators or to change the language generator without impacting the rest of the system.  **[Bad] Impacts complexity**  The introduction of an additional transformation layer increases the system’s complexity.  **[Bad] Can impact performance**  As is obvious, the intermediate language generation approach uses multiple transformations/processing layers on the workflow model. This can impact system performance in terms of an increase in response time. | In this approach the workflow language is directly generated by the tool from the workflow graphical model (which is in JSON format).  [**Bad**] **Reduces extensibility**  Here the extensibility capability of the tool to support multiple workflow languages and ease to move from one language to another is severely inhibited. The direct generation approach ties the workflow tool with the actual workflow language used by SORASCS. If SORASCS decides to go from BPEL to something else then the “workflow generator” component will have to be re-written. |

**Decision: Generate an intermediate language.**

After weighing the two approaches, the team along with the client decided to go with the intermediate language option. However, a decision still needs to be taken about what the format of this language will be. Currently, the team has conducted experiments with “Automap script,” which is an XML format language generated by the CASOS tool Automap [[15](#CAS10)]. However, in the future there is the possibility to have Acme[[16](#Arc09)] as an intermediate language.

in *Section*  shows a schematic diagram of direct generation and intermediate language generation.

## Analysis Plug-ins as Web Services

Analyses are a way to analyze the properties of a given workflow. Possible examples include: “How long will it take for this workflow to execute?”, “Do all the services used in this workflow meet the security policies?”, and “Are all the services used in this workflow in the correct order?”. The tool must support a way for users to develop and “plug in” their own analysis algorithms, but the exact form of this plug-in framework was an architectural decision that the team had to make. As described in *Section* , extensibility is the main architectural driver for this decision and web-based graphical framework is the technical constraint. Given the web-based graphical framework that the team was evaluating (WireIt), it was important to find out how well, and with what mechanisms, WireIt would support this extension point.

### Trade-Offs

Below are the set of questions that were answered via experiments and brainstorming sessions to make the decision.

**Question: What will the analyses be developed as?**

|  |  |
| --- | --- |
| **Web Service** | **Primitive Language Only** |
| **[Good] Promotes flexibility.**  Web services give the analysis developer flexibility in the implementation language.  **[Good]** **Promotes standardization.**  Because CASOS tools capabilities are also deployed as web-services, deploying an analysis will be similar to deploying a tool service.  **[Good]** **End-user may benefit by having access to more choices of analyses as they are available as web-service.**  **[Bad]**  **Inhibits Performance.**  Web service execution is comparably slower than doing primitive local call execution. | **[Bad]** **Inhibits flexibility.**  Analyses developer is tied or constrained to our language. |
| **[Neutral]** **End-user is not impacted whether this is web-service or not.** | |

**Questions:**

1. **Where will the registry reside when an analysis is added and where will it be retrieved from?**
2. **Where does execution of the analyses take place?**

|  |  |
| --- | --- |
| **On SORASCS** | **On Middle Tier** |
| **[Good] Promotes design and implementation reusability.**  If on SORASCS, analyses can be added to the existing web-service registry.  **[Good] Promotes design and implementation reusability.**  The end user can have the same access control policies as CASOS services to make analyses services also available and shared to other users. | **[Bad]** **Inhibits maintainability.**  If on middle tier, and the implementation is the same as the SORASCS service registry, then this will have to be maintained in two places.  **[Bad] Adds complexity.**  The FACEO5 team will have to develop user access control if it is desired to make analyses sharable. |
| **[Neutral] For analyses developer there is no impact.**  **[Neutral] No impact for execution of analyses for an analyses end-user** | |

**Question: Is this plug-in extensibility supported by Wire IT web graphical framework?**

Yes. Based on the experimentation using WireIt to access the web services from SORASCS we decided to design the analyses as a web-service. Hence, the WireIt graphical framework supports extensibility to add and retrieve analyses during run-time.

**Question:** **How are user permissions controlled (who can see what analyses)? Are all analyses seen by all users?**

We can use the same SORASCS access control mechanism where admin controls user access of other services.

### Analysis

Based on the above trade-off discussion, following are the decisions that will address our architectural driver concerns:

* Analyses will be web-service.
* Analyses registry and execution will reside on SORASCS, and use the existing service registry and execution framework on SORASCS for analyses service discovery and execution.
* This plug-in extensibility capability is supported by the WireIt web graphical framework.

There is forward work on answering some of the detailed questions regarding analysis plug-in framework, but they will not impact our architecture choice above. An example of one of these questions is: how do we distinguish analyses from the other services that are deployed? Two possible answers are tagging and type of service - analyses vs. user defined deployed workflow (service). Two other questions are: What interface and data do we require from analyses and provide to analyses? What data do we expose to/from SORASCS?

## Google Gears

As listed in *Section 2* , an analyst should be able to perform workflow construction and save operations independent of whether SORASCS is offline or online. Also, choosing a 3-tier architectural style introduced inhibiting performance characteristics to the overall system. To address this problem, there was a need to work on a strategy to have “some” data available locally. This will allow the analyst to continue the workflow construction activities during SORASCS unavailability and provide a faster user response while loading the tool with a growing number of services. However, the addition of a local data access layer brings in the dimension of increased “complexity” to the system because now we need to decide what data needs to be present locally, and we also need to periodically synchronize the local data store with the data on the server.

Given this, the team performed research into what options are available to cache the service and workflow data locally on the user’s machine. Possible web-based frameworks were looked into that can provide an API to store and retrieve data locally. During this exercise, the Google Gears framework [[5](#Goo08)] was found to support this need.

Following are the results of the high-level analysis of Google Gears experimentation using WireIt and the SORASCS interface:

* **[Good] Ease of use.**

Provides an API to perform CRUD operations on local data. Local data is stored in flat file based SQLite database.

* **[Good] Promotes Performance**

For the set of workflow construction steps where the “data” is cached and available locally, the browser does not need to make the call to the server. This would promote performance in the “workflow construction” and “workflow save” scenarios.

* **[Good] & [Bad] Discontinued Google Gears development effort to promote HTML5 standardization.**

It was found that the Google Gears development was going to stop and its capabilities were going to be transitioned to the HTML5 standardization. [[17](#Goo10)] This is good in a sense that there is going to be a standardized way to add capabilities to browsers to work offline. However, it is bad in a sense that this might not be ready in the time that the project needs to have it running; also, the browser support with HTML5 is right now limited to Chrome, IE8 or above, and Firefox.

As a result of doing a prototype to store and retrieve data locally from a browser environment, we found that Google Gears is a possible framework for local data access. However, the usage of Google Gears is left as an open option based on following things:

* Evolution of HTML5 standardization.
* Analysis of what data will be stored locally and what data will be stored remotely, along with the frequency of data synchronization.
* Decision on what steps of workflow orchestration should be allowed when SORASCS is unavailable.

# Appendix A: Technical Glossary

## Data Formats

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **References** |
| XML | XML (Extensible Markup Language) is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C and several other related specifications; all are fee-free open standards.  XML’s design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML’s design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.  There are many programming interfaces that software developers may use to access XML data, and several schema systems designed to aid in the definition of XML-based languages. | http://en.wikipedia.org/wiki/XML |
| JSON | JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.  JSON is built on two structures:  A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.  An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence. | <http://www.json.org> |
| WSDL | WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME. | http://www.w3.org/TR/wsdl http://en.wikipedia.org/wiki/Web\_Services\_Description\_Language |

## Protocols

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Term** | | **Description** | **References** | |
| HTTP | The Hypertext Transfer Protocol (HTTP) is an Application Layer protocol for distributed, collaborative, hypermedia information systems.  HTTP is a request-response standard typical of client-server computing. In HTTP, web browsers or spiders typically act as clients, while an application running on the computer hosting the web site acts as a server. The client, which submits HTTP requests, is also referred to as the user agent. The responding server, which stores or creates resources such as HTML files and images, may be called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels. | | http://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol |
| SOAP | SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on eXtensible Markup Language (XML) as its message format, and usually relies on other Application Layer protocols (most notably Remote Procedure Call (RPC) and HTTP) for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built. | | http://en.wikipedia.org/wiki/SOAP\_(protocol) |

## Languages

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **References** |
| JavaScript | JavaScript is an object-oriented scripting language used to enable programmatic access to computational objects within a host environment. Although also used in other applications, it is primarily used in the form of client-side JavaScript, implemented as part of a web browser, providing enhanced user interfaces and dynamic websites. JavaScript is a dialect of the ECMAScript standard and is characterized as a dynamic, weakly typed, prototype-based language with first-class functions. JavaScript was influenced by many languages and was designed to look like Java, but to be easier for non-programmers to work with. | http://en.wikipedia.org/wiki/JavaScript |
| BPEL | Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL) is an OASIS standard executable language for specifying interactions with Web Services. Processes in Business Process Execution Language export and import information by using Web Service interfaces exclusively. BPEL is an Orchestration language, not a choreography language (see Web Service Choreography). The primary difference between orchestration and choreography is executability and control. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. | http://en.wikipedia.org/wiki/BPEL |

## Message Passing

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **References** |
| PiggyBack Technique | Here the server, having an update to send, waits for the next time the browser asks it a question and… pounce, sends the answer and the update. This technique is well understood in long term relationships and is usually preceded by ‘and another thing.’ You know, you ask what you thought was a straight forward question and get a lot more than you bargained for. | http://ajaxian.com/archives/reverse-ajax-with-dwr |
| Comet, long lived Http, or the slow load technique: | Are all names for the same thing. As already mentioned, the server has to wait for the browser to make contact. But this technique allows the server to start answering the browser’s request for information very slowly. Extremely slowly. Actually in the same way I used to answer my French teachers at school, it starts the reply but never actually finishes. This allows the server to keep a communications channel open (unlike me and my French teacher) to pass down additional information when the time comes. The closest we currently get to a server push. See Alex Russell’s article for the coining of the phrase and outline of definition of Comet. See Bryce Nesbitt’s for a brief description and simple demo of slow-load. | http://ajaxian.com/archives/reverse-ajax-with-dwr |
| Polling | This is where the browser makes a request of the server at regular and frequent intervals, say every 3 seconds to see if there has been an update to the page. To illustrate imagine a 5 year old (or a donkey) in the back of the car shouting ‘are we there yet?’ every few seconds and you get the picture. | http://ajaxian.com/archives/reverse-ajax-with-dwr |

## Other

|  |  |  |
| --- | --- | --- |
| **Term** | **Description** | **References** |
| SOA |  | http://www.service-architecture.com/web-services/articles/service-oriented\_architecture\_soa\_definition.html |
| Webservice | Web services are typically application programming interfaces (API) or web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.  In common usage the term refers to clients and servers that communicate over the Hypertext Transfer Protocol (HTTP) protocol used on the web. | http://en.wikipedia.org/wiki/Web\_servics |
| DWR | DWR will generate the JavaScript to allow web browsers to securely call into Java code almost as if it was running locally. It can marshal virtually any data including collections, POJOs, XML and binary data like images and PDF files. All that is required is a security policy that defines what is allowed.  With Reverse Ajax, DWR allows Java code running on a server to use client side APIs to publish updates to arbitrary groups of browsers. This allows interaction 2 ways - browser calling server and server calling browser. DWR supports Comet, Polling and Piggyback (sending data in with normal requests) as ways to publish to browsers. | http://directwebremoting.org/dwr/index.html |
| Google Gears | Gears is an open source project that enables more powerful web applications, by adding new features to your web browser:  Let web applications interact naturally with your desktop   Store data locally in a fully-searchable database   Run JavaScript in the background to improve performance.   Gears is a plug-in that extends your browser to create a richer platform for web applications. For example, webmasters can use Gears on their websites to let users access information offline or provide you with content based on your geographical location. To install Gears, visit http://gears.google.com   Gears was designed to be used on both Google and non-Google sites. A number of web applications currently make use of Gears, including two Google products: Google Reader and Google Docs. Additionally, Zoho and Remember the Milk have been using Gears since its original launch. If you're running Windows Mobile on your cellphone, Picasa Web Albums also makes use of Gears. | http://gears.google.com/ |
| AJAX | AJAX = Asynchronous JavaScript and XML.  AJAX is not a new programming language, but a new way to use existing standards.  AJAX is the art of exchanging data with a server, and update parts of a web page - without reloading the whole page. | http://www.w3schools.com/Ajax/Default.Asp http://en.wikipedia.org/wiki/Ajax\_(programming) |
| Reverse AJAX | Reverse Ajax refers to an Ajax design pattern that uses long-lived HTTP connections to enable low-latency communication between a web server and a browser. Basically it is a way of sending data from client to server and a mechanism for pushing server data back to the browser.  This server–client communication takes one of two forms:  Client polling: the client repeatedly queries (polls) the server and waits for an answer.  Server pushing: a connection between a server and client is kept open and the server sends data when available.  Reverse Ajax describes the implementation of either of these models, or a combination of both. The design pattern is also known as Ajax Push, Full Duplex Ajax and Streaming Ajax. Less traffic is generated with Reverse Ajax and messages are transferred with less delay (low-latency). | http://en.wikipedia.org/wiki/Reverse\_Ajax |

# Appendix B: Complete Utility Tree

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quality Attribute** | **Attribute Refinement** | **Scenario** | **Stakeholder Priority** | **Perceived Difficulty** |
| Performance | Backend interaction | The tool should be able to load and save a workflow of 50 services in no more than five seconds. | M | H |
| Extensibility | Analysis Extension Framework | A SORASCS developer has a new analysis to use on workflows. Given that they have developed the algorithms for this analysis, they should be able to write and register the new workflow analysis in the tool in less than one week. | M | H |
|  | Palette Extension Framework | An analyst has a new workflow service organization plug-in that they want to add to the services palette. Given they have developed the logic for the plug in, they should be able to write and register a new service organization plug in with the tool in less one week. | M | M |
| Modifiability | SORASCS Workflow description language modification | A SORASCS developer wants to change the workflow description language which is generated from the intermediate language. They can develop and test the new language generation logic without changing/affecting the existing front end implementation of the workflow tool. | M | M |
| Scalability | Fast load and backend interaction. | An analyst launches the tool. They can see the organized palette with components that are only available for the analyst in no more than 10 seconds. | M | H |
| Usability | Easy packaging and sharing | Given that an analyst has a syntactically valid workflow that they want to share with other users as a service, the workflow can be packaged, parameterized, have meta-data entered, deployed, and available for others users to use within one hour. | H | M |
|  | Sensible organization | SORASCS has 1000 services registered for it. An analyst should be able to locate a service to add to a workflow in no more than five seconds. | H | M |
|  | Easy re-execution | An analyst wants to re-execute a workflow with a new dataset. He or she should be able to do this with less than ten seconds of effort. | H | L |
|  | Feedback | An analyst should get feedback of “SORASCS not available” within 3 seconds during any of the workflow lifecycle stages in case of SORASCS outage. | M | L |
|  | Easy to install | The tool should be installable with less than 3 steps, and it should be apparent to the analyst whether his or her system's requirements meet the minimum requirements. If an analyst tries to install the workflow tool on a system that is not compatible, the installer recommends and can optionally download all the prerequisites. | L | L |
|  |  |  |  |  |
|  | Offer suggestions | An analyst tries to load a workflow for which they do not have access to some of the services or files. The system will suggest comparable services and will provide advice (appropriate to the analyst's experience) about the impact of the substitution. When providing advice about substituting or tailoring services, that advice is understood by the analyst 99% of the time. | L | M |
|  | Easy to learn | An analyst who is experienced with the current CASOS tools wants to construct a workflow from scratch using services that currently exist. He or she is able to construct and test the workflow with a small data set within two hours. | H | H |
|  | Non-blocking | The tool interacts with SORASCS server at various points of time. At every point of time during server interaction, UI should be responsive within less than 2 seconds. | H | H |
| Availability | Tool usage with local SORASCS | Users who have SORASCS installed locally should be able to use the workflow tool without connecting to the remote SORASCS serve. | L | L |
| Security | Secured communication | When the tool communicates over the network, the information given by the analyst is securely protected and not picked up by other person via network. | M | L |

# Appendix D: DWR Technical Notes

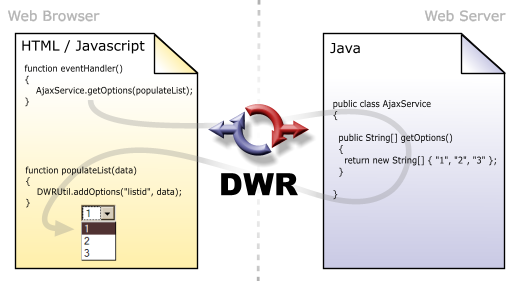
Direct Web Remoting (DWR) is a Java open source library that helps developers write web applications that include Ajax technology. It allows code in a web browser to use Java functions running on a web server as if those functions were within the browser.

DWR consists of two main parts:

* A Java Servlet running on the server that processes requests and sends responses back to the browser.
* JavaScript running in the browser that sends requests and can dynamically update the webpage.

DWR works by dynamically generating JavaScript based on Java classes.

Here is a cartoon from DWR web site explaining its functionality.

****

Here the JavaScript code is dynamically generated from the Java code by DWR. DWR takes care of marshalling and un-marshalling the data types from JavaScript to Java and vice versa.

More details can be found DWR’s web site [[12](#Dir10)].

# Appendix F: Links to Other Resources

1. Sequence diagrams

<http://msesrv2a.mse.cs.cmu.edu:7080/download/attachments/2064473/SequenceDiagrams.pdf>

1. Architectural drivers to key challenges and experiments mapping

<http://msesrv2a.mse.cs.cmu.edu:7080/display/archwiki/Architectural+Drivers+To+Key+Challenges+To+Experiments+Mapping>

1. Graphical framework investigation

<http://msesrv2a.mse.cs.cmu.edu:7080/display/archwiki/Graphical+framework+investigation>

# References

x

|  |  |
| --- | --- |
| [1] | David Garlan, Kathleen M. Carley, Bradley Schmerl, Michael Bigrigg, and Orieta Celiku, "Using Service-Oriented Architectures for Socio-Cultural Analysis," in *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering*, Boston, USA, 2009. |
| [2] | IBM. (2007, Feb) Business Process Execution Language for Web Services version 1.1. [Online]. <http://www.ibm.com/developerworks/library/specification/ws-bpel/> |
| [3] | The Apache Software Foundation. (2010) Apache ODE. [Online]. <http://ode.apache.org/> |
| [4] | The Apache Software Foundation. (2010) Apache Tomcat. [Online]. <http://tomcat.apache.org/> |
| [5] | Google. (2008) Gears. [Online]. <http://gears.google.com/> |
| [6] | SQLite. SQLite Home Page. [Online]. <http://sqlite.org/> |
| [7] | Oracle Corporation. (2010) JDK 6 Java Database Connectivity (JDBC)-related APIs & Developer Guides. [Online]. <http://java.sun.com/javase/6/docs/technotes/guides/jdbc/> |
| [8] | WireIt. (2010) WireIt - a Javascript Wiring Library. [Online]. <http://javascript.neyric.com/wireit/> |
| [9] | Andreas Herz. (2010) Open-jACOB Draw2D. [Online]. <http://www.draw2d.org/draw2d/> |
| [10] | Yahoo! Inc. (2010) Pipes: Rewire the web. [Online]. <http://pipes.yahoo.com/pipes/> |
| [11] | (2010) JSON. [Online]. <http://www.json.org/> |
| [12] | Direct Web Remoting. (2010) DWR - Easy Ajax for JAVA. [Online]. <http://directwebremoting.org/dwr/index.html> |
| [13] | Google. (2010) Google Web Toolkit Overview. [Online]. <http://code.google.com/webtoolkit/overview.html> |
| [14] | Mozilla. (2010) Firebug. [Online]. <http://getfirebug.com/> |
| [15] | CASOS. (2010) Automap: Project. [Online]. <http://www.casos.cs.cmu.edu/projects/automap/> |
| [16] | Architecture Based Languages and Environments group. (2009) Acme - The Acme Architectural Description Language and Design Environment. [Online]. <http://www.cs.cmu.edu/~acme/> |
| [17] | Google Inc. (2010, February) Gears API Blog: Hello HTML5. [Online]. <http://gearsblog.blogspot.com/2010/02/hello-html5.html> |
| [18] | David Garlan and Bradley Schmerl, "AcmeStudio: Supporting Style-Centered Architecture Development," in *International Conference on Software Engineering*, Edinburgh, Scotland, 2004. |

x