Software Architecture

Front-End, Microservices, and Backend

William M. Mongan

Microservices and Service-Oriented Architectures

Service Oriented Computing (SOC) is a movement towards an implementation-independent architecture for distributed computing.

Rather than an object-based design and functionality, Service Oriented Architecture (SOA) raises the level of abstraction towards higher level business logic.

A Typical Scenario

A small company is currently using separate in-house proprietary systems to process orders, charge credit cards, check inventory and ship products.

Data is exchanged from one department to the other via hand-written e-mails, though the inventory and shipping departments were recently integrated and sharing messages using a proprietary format over TCP.

Why SOA

In this example, departments only expose the highest level of business logic that each needs to share.

For example, the shipping department need only know that the credit card transaction was approved and the shipping address provided by the customer.

Bottom Up vs. Top Down Design

Although SOA allows for a bottom-up design amicable to the integration of legacy systems, the real benefit is realized from a top-down, implementation-independent approach.

Bottom Up vs. Top Down Design

By using schemas to define abstract data types and to define business-logic operations in terms of these abstract types, a complete interface is derived for a system that naturally hides implementation details and business secrets.

The design hides the implementation: Services only provide their public interfaces to one another.

So What?

Imagine services as a pipeline, or assembly line, through which business logic is realized.

If a service (i.e. the shipping department) was outsourced to a third party, only the service contract and the surrounding clients that invoke it need to be changed.

- The new shipping company does not want to disclose private information such as its sub-contractors and charges.
- Only the invocation requirements, message format, and data parameters are exposed by the contract.

Advantages to SOA

Services are discoverable and dynamically bound;

Services are self-contained and modular;

Services stress interoperability;

Services are loosely coupled;

Services have a network-addressable interface;

Services have coarse-grained interfaces;

Services are location transparent;

Services are composable;

Service-oriented architecture supports self-healing.

Service Composition

Because services are a primitive unit in SOA, they can be composed just as objects are in the OO paradigm to create new applications according to business processes.

Unlike objects, services are also registered with a name service that can be searched.

In this way, services can be dynamically found, bound, and consumed at runtime.

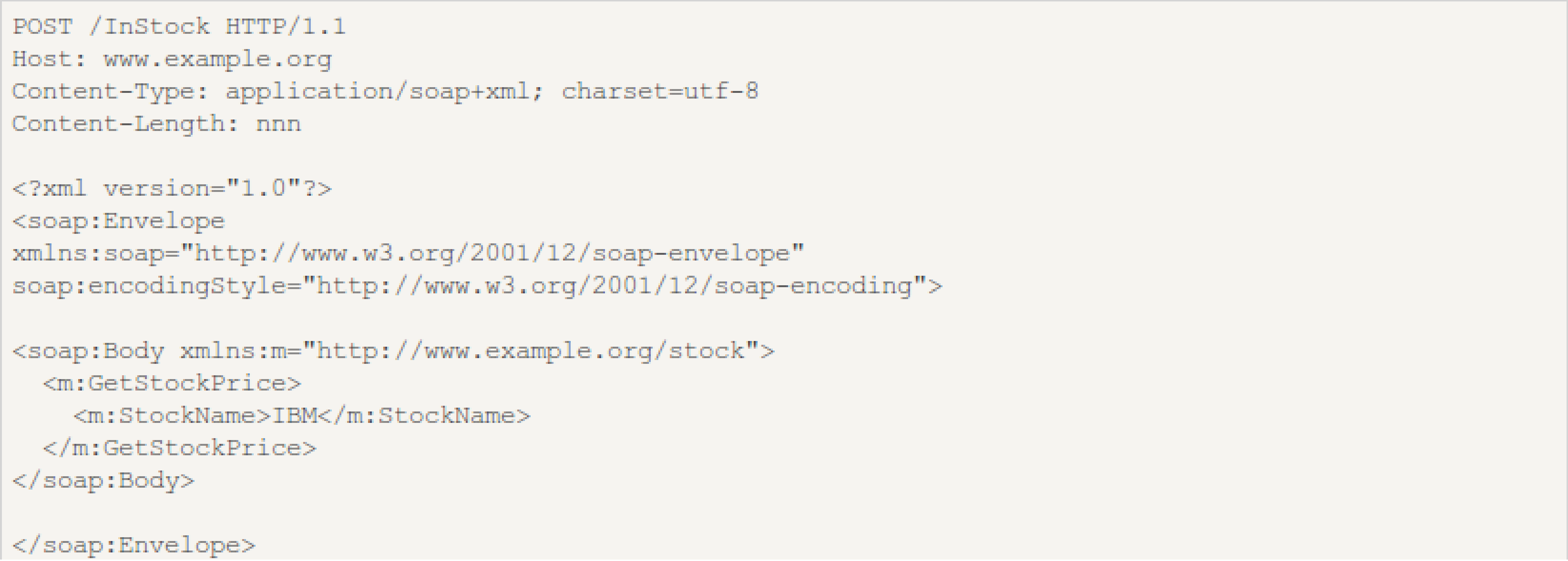
An interesting and open problem is to dynamically bind to services based on a semantic query.

Service Flexibility

This flexibility facilitates a heterogeneous and asynchronous service environment.

Of course, one requirement is that the decision must be made and agreed upon by the service and caller.

Example SOAP Request and Response

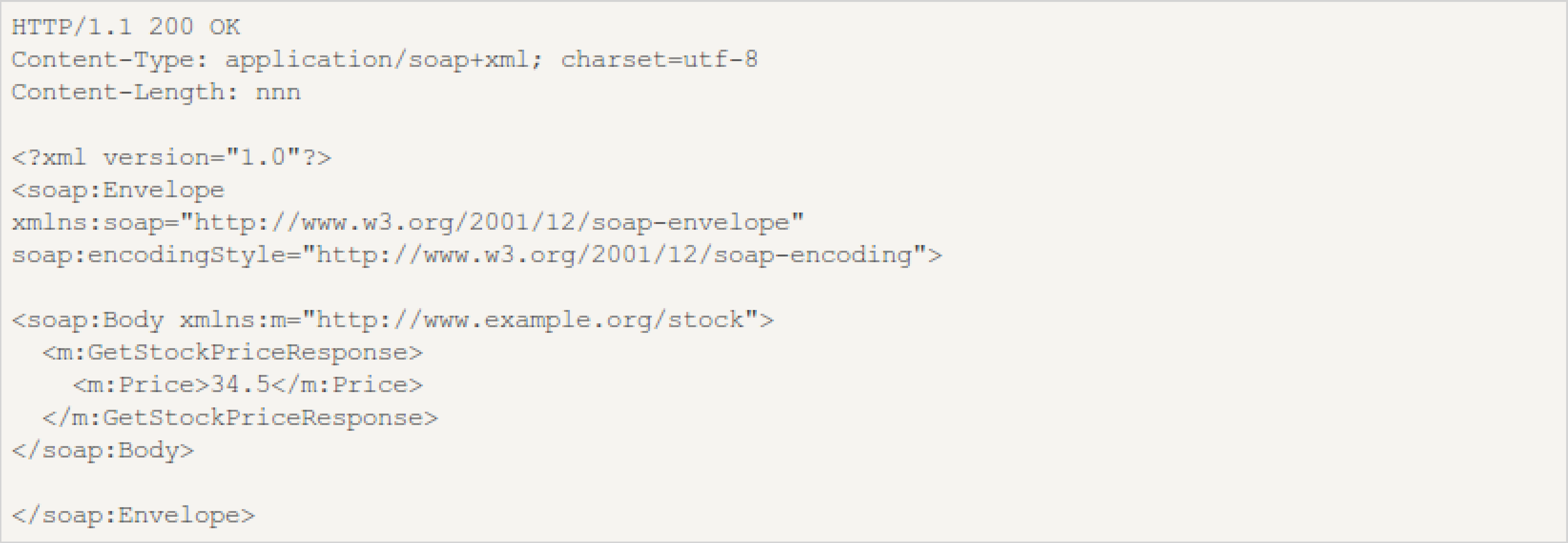


```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

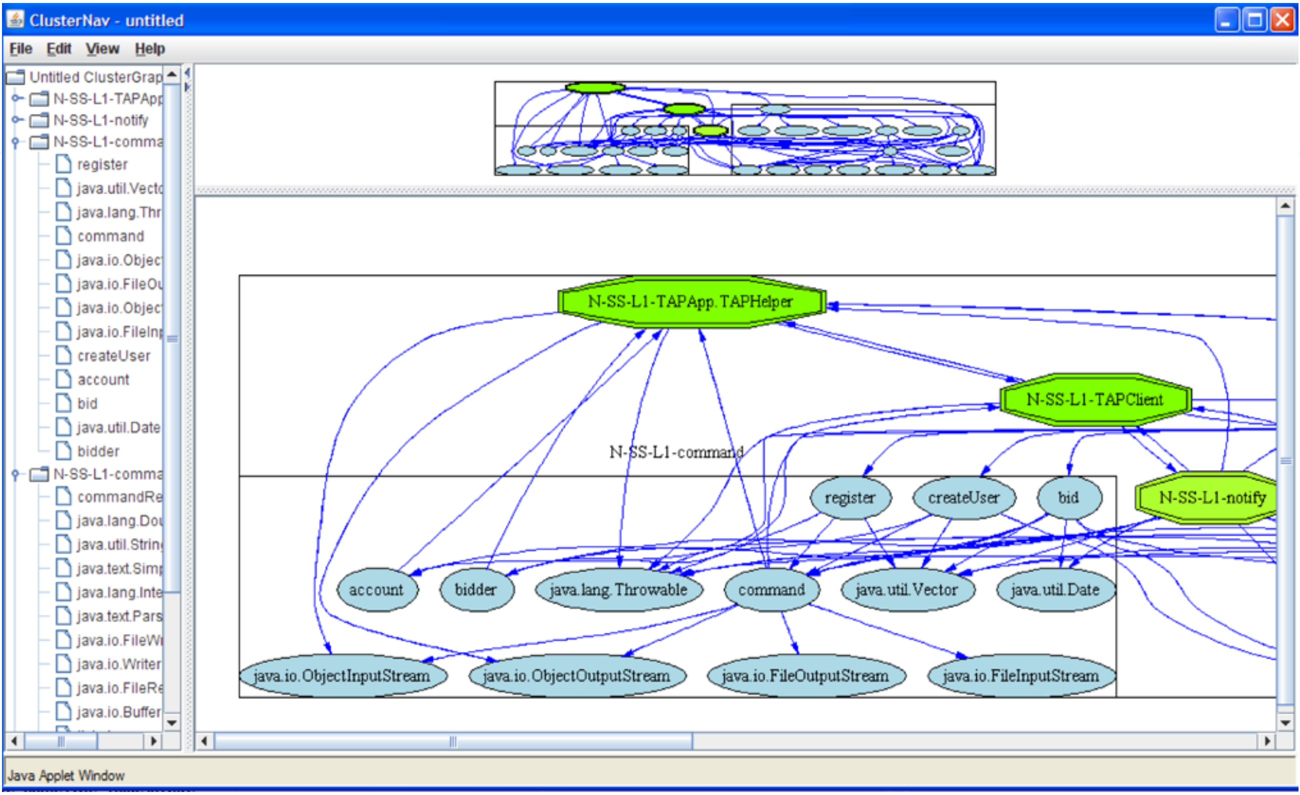


```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```



CASE STUDY: REPORTAL

Case Study: REportal

REportal is a service-based reverse engineering portal.

Users may upload code to REportal and perform RE analysis, without needing to install, configure and run individual tools.

Brief History

REportal 1.0 was based on Java Servlets, but the presentation layer was tightly coupled to the tools.

The tools quickly became obsolete, and others simply didn't work, hindering the functionality.

Browser wars were a big problem during this time as well.

REportal 2.0

REportal 2.0 is based on web services. This was chosen because the architecture decouples the interface from the tools.

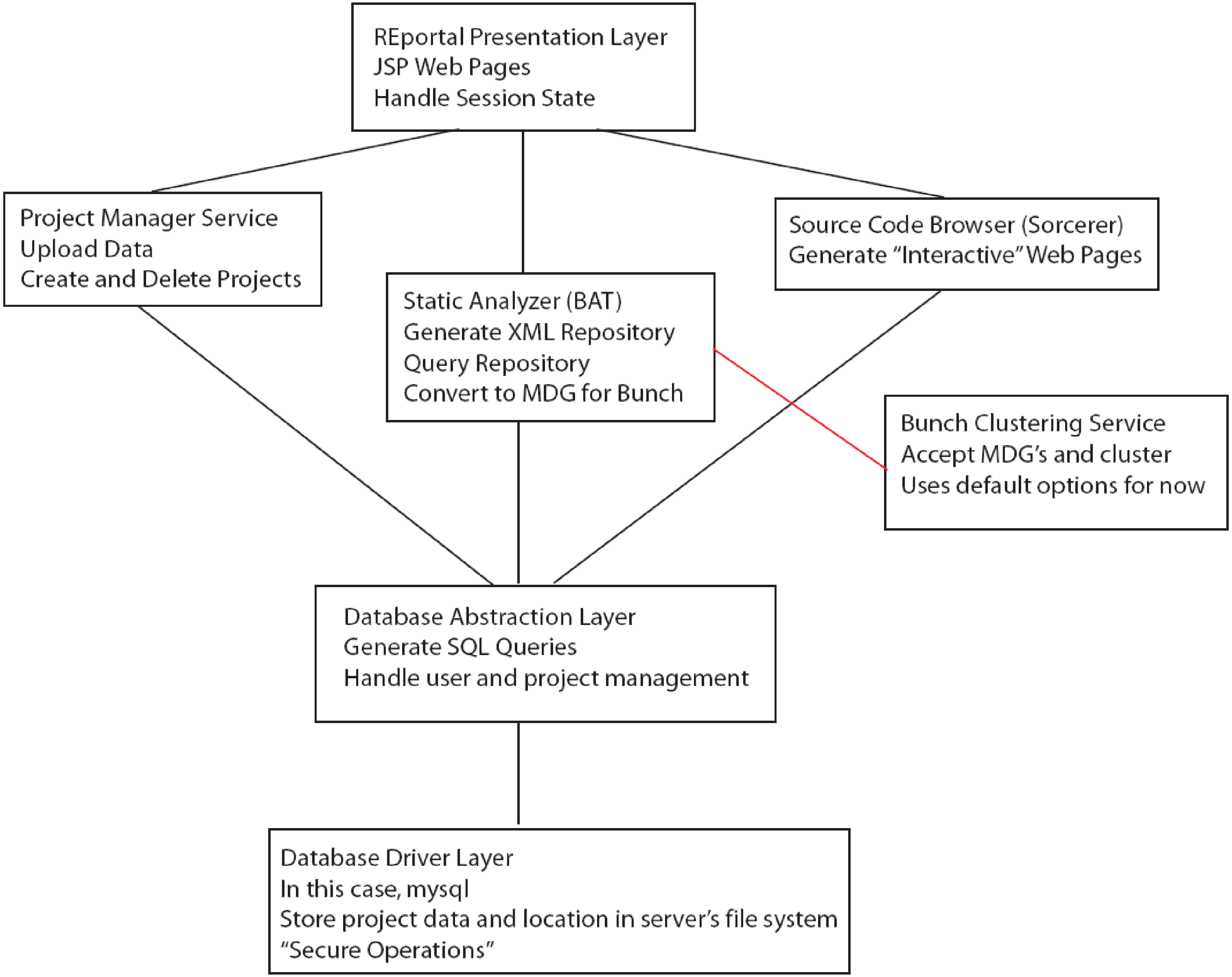
Relationships between tools are based on data via message passing in XML.

This makes it easy to

Maintain existing tools.

Add new (legacy) tools by simply turning them into a service.

Take advantage of SOA security, automatic binding, distributed computing, and other features.



REportal Presentation Layer
JSP Web Pages
Handle Session State

Project Manager Service
Upload Data
Create and Delete Projects

Static Analyzer (BAT)
Generate XML Repository
Query Repository
Convert to MDG for Bunch

Source Code Browser (Sorcerer)
Generate "Interactive" Web Pages

Bunch Clustering Service
Accept MDG's and cluster
Uses default options for now

Database Abstraction Layer
Generate SQL Queries
Handle user and project management

Database Driver Layer
In this case, mysql
Store project data and location in server's file system
"Secure Operations"

The Tools (Services)

REportal Application Layer

Bunch Wrapper

Static Analyzer

Source Code Browser

Text Search

Aspect Instrumentation

Project Manager

Database Layer

The Tools

REportal Application Layer

This is the presentation layer, where JSP pages reside. The JSP pages invoke services to render functionality.

Bunch Wrapper

This is used by several services, whenever graphical data or an MDG is produced. Bunch Wrapper clusters the graph and returns a new graph.

The Tools

Static Analyzer

This is based on the BAT Static Analyzer for Java 1.5.

Given Java class file(s), BAT creates an XML repository with source code relationships that exist between the entities.

- This needs to be improved to a database model for scalability.
- REportal queries the repository via XQuery and XSLT.

The Tools

Source Code Browser

This is based on the Sorcerer source code browser tool. It provides a cross-referenced source index.

Currently, REportal downloads a zip file of web pages to the user.

- Ultimately, using Ajax, we will display this content on the fly, rendering a seamless user interface!

Text Search

Grep

The Tools

Aspect Instrumentation

Using aspects, it is possible to instrument code to trace function calls.

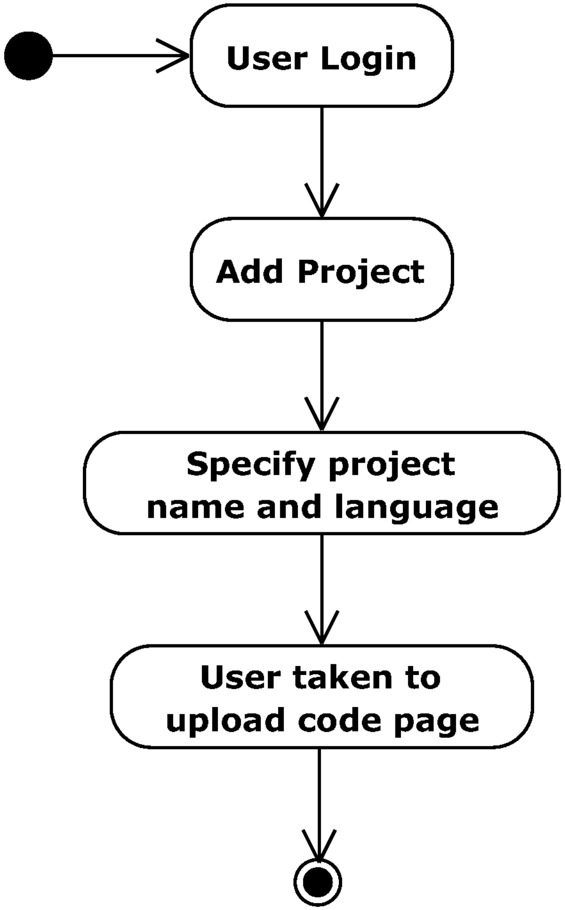
Doing this, we yield an MDG graph that can be viewed or clustered to show runtime slices.

Project Manager and Database Layer

Using a database, tracks user logins and projects, including their location on the file system.

They don’t have to reside on the same machine, but the database provides absolute file paths on the Project Manager service’s file system.

Sample Use Case: Adding a Project

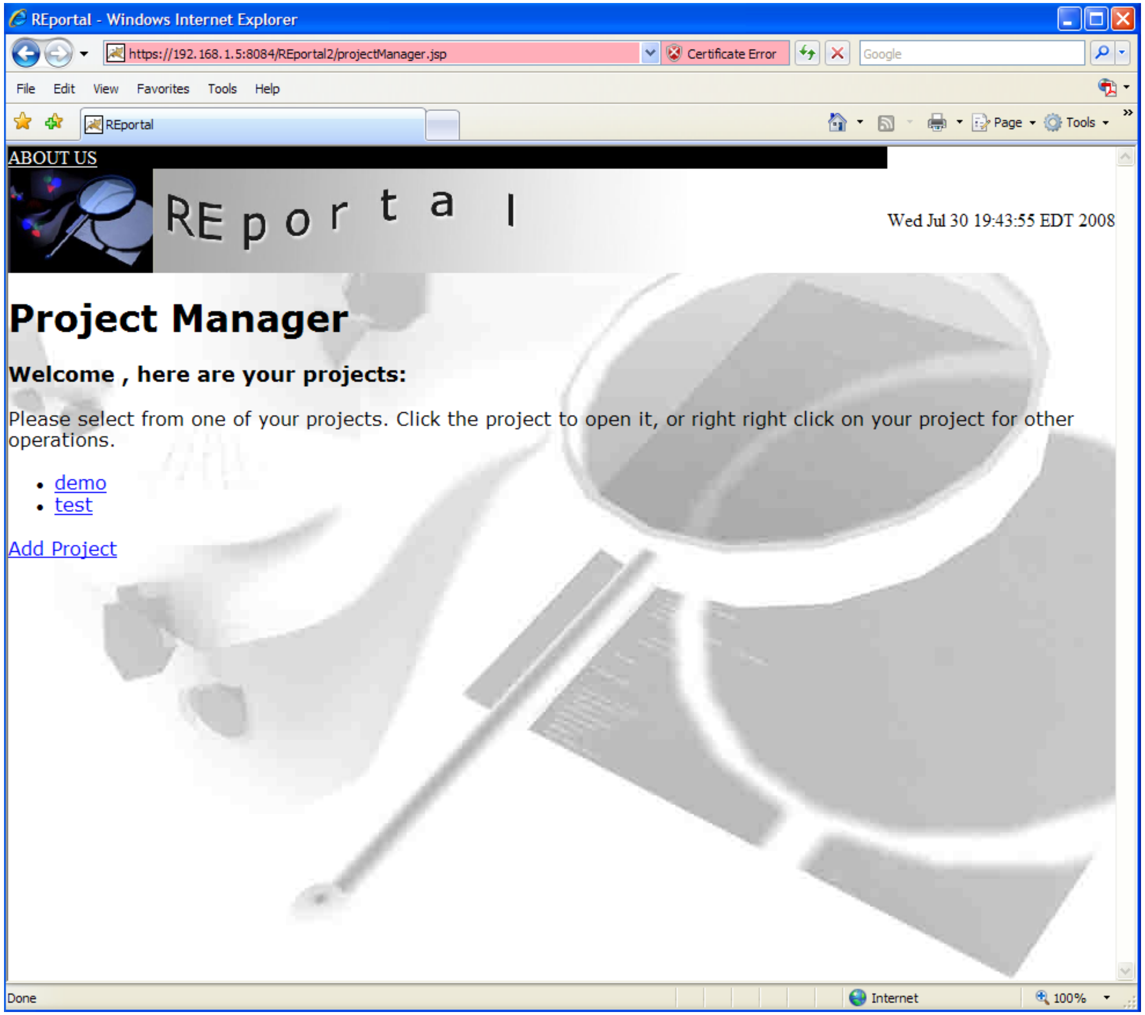


User Login

Add Project

Specify project name and language

User taken to upload code page



Wed Jul 30 19:43:55 EDT 2008

Project Manager

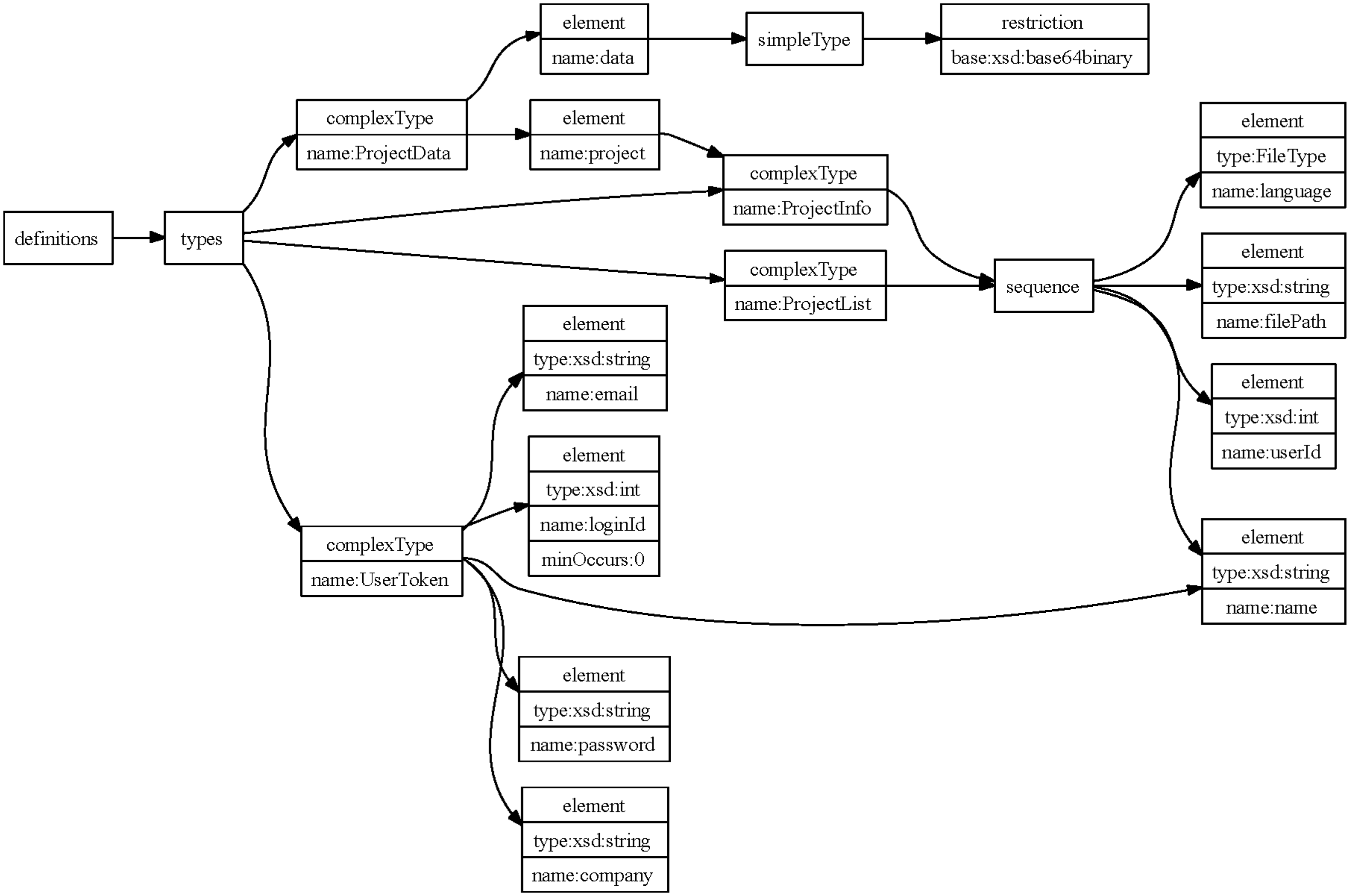
Welcome , here are your projects:

Please select from one of your projects. Click the project to open it, or right right click on your project for other operations.

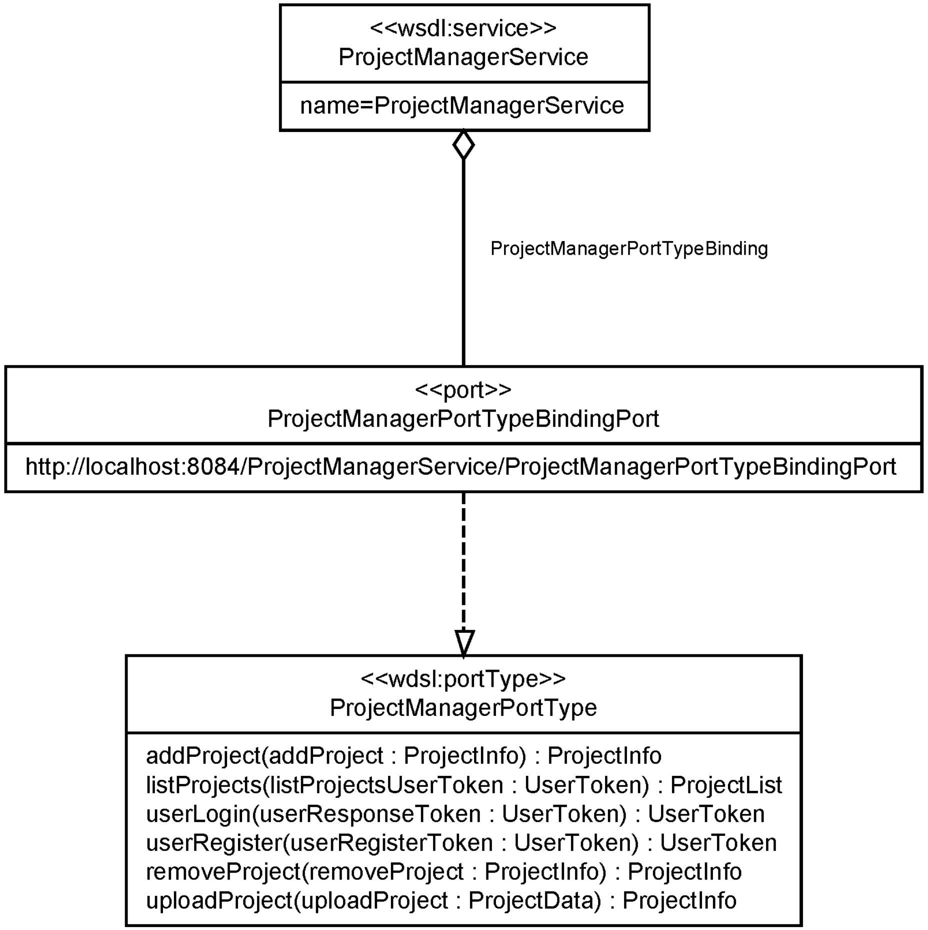
- demo
- test

Add Project

Sample Service Definition: Adding a Project



Sample Service Interface: Adding a Project



<<wsdl:service>>
ProjectManagerService

name=ProjectManagerService

ProjectManagerPortTypeBinding

<<port>>
ProjectManagerPortTypeBindingPort

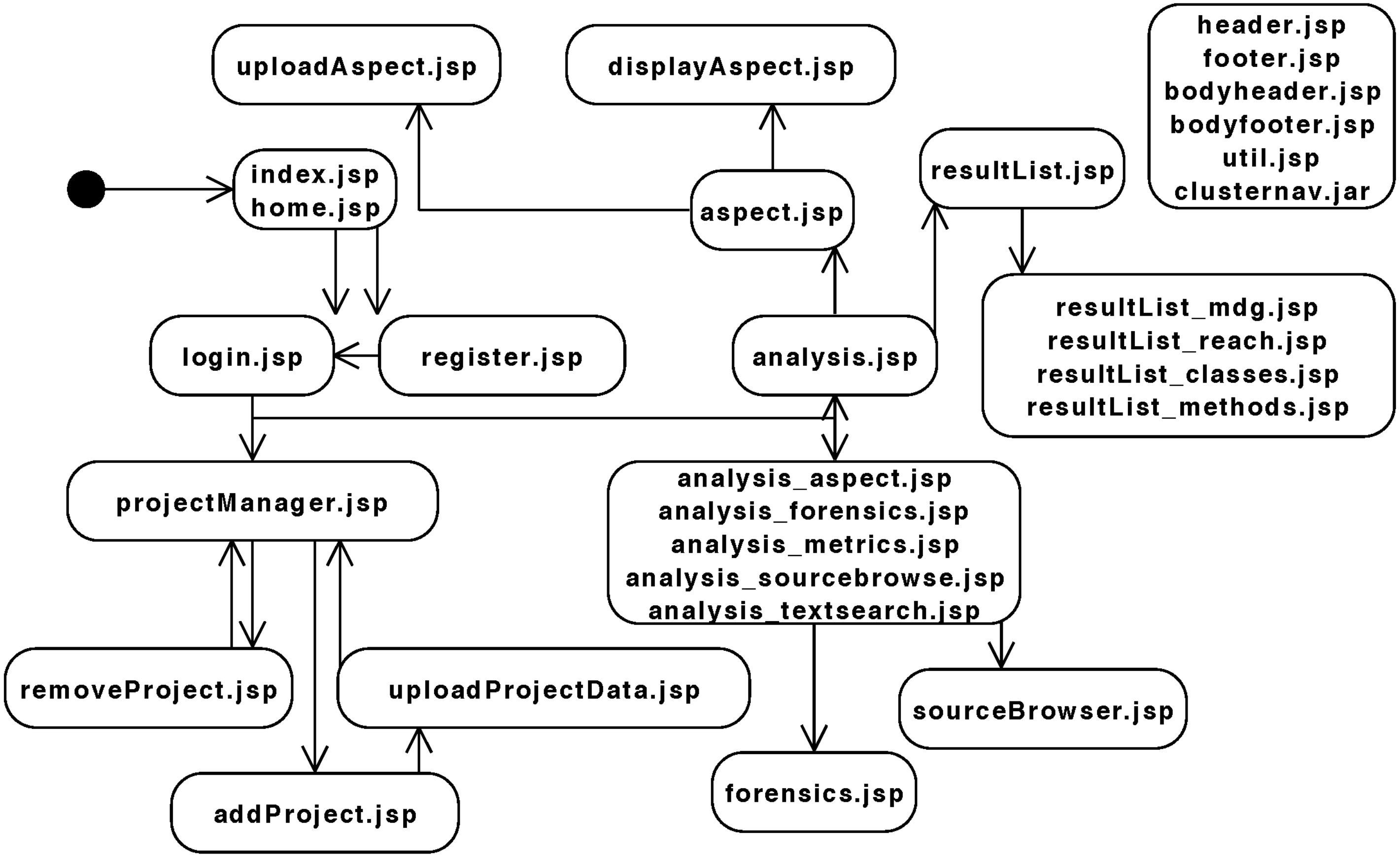
http://localhost:8084/ProjectManagerService/ProjectManagerPortTypeBindingPort

<<wdsl:portType>>
ProjectManagerPortType

addProject(addProject : ProjectInfo) : ProjectInfo
listProjects(listProjectsUserToken : UserToken) : ProjectList
userLogin(userResponseToken : UserToken) : UserToken
userRegister(userRegisterToken : UserToken) : UserToken
removeProject(removeProject : ProjectInfo) : ProjectInfo
uploadProject(uploadProject : ProjectData) : ProjectInfo

PROJECT MANAGER SERVICE

Front-End Design and Presentation



uploadAspect.jsp

displayAspect.jsp

header.jsp
footer.jsp
bodyheader.jsp
bodyfooter.jsp
util.jsp
clusternav.jar

index.jsp
home.jsp

aspect.jsp

resultList.jsp

login.jsp

register.jsp

analysis.jsp

resultList_mdg.jsp
resultList_reach.jsp
resultList_classes.jsp
resultList_methods.jsp

projectManager.jsp

analysis_aspect.jsp
analysis_forensics.jsp
analysis_metrics.jsp
analysis_sourcebrowse.jsp
analysis_textsearch.jsp

removeProject.jsp

uploadProjectData.jsp

sourceBrowser.jsp

forensics.jsp

addProject.jsp

CASE STUDY: THE IOT SENSOR FRAMEWORK

Case Study: IoT Sensor Framework

This software suite contains scripts to collect and store IoT sensor data, such as RFID tag information using an Impinj Speedway RFID reader.

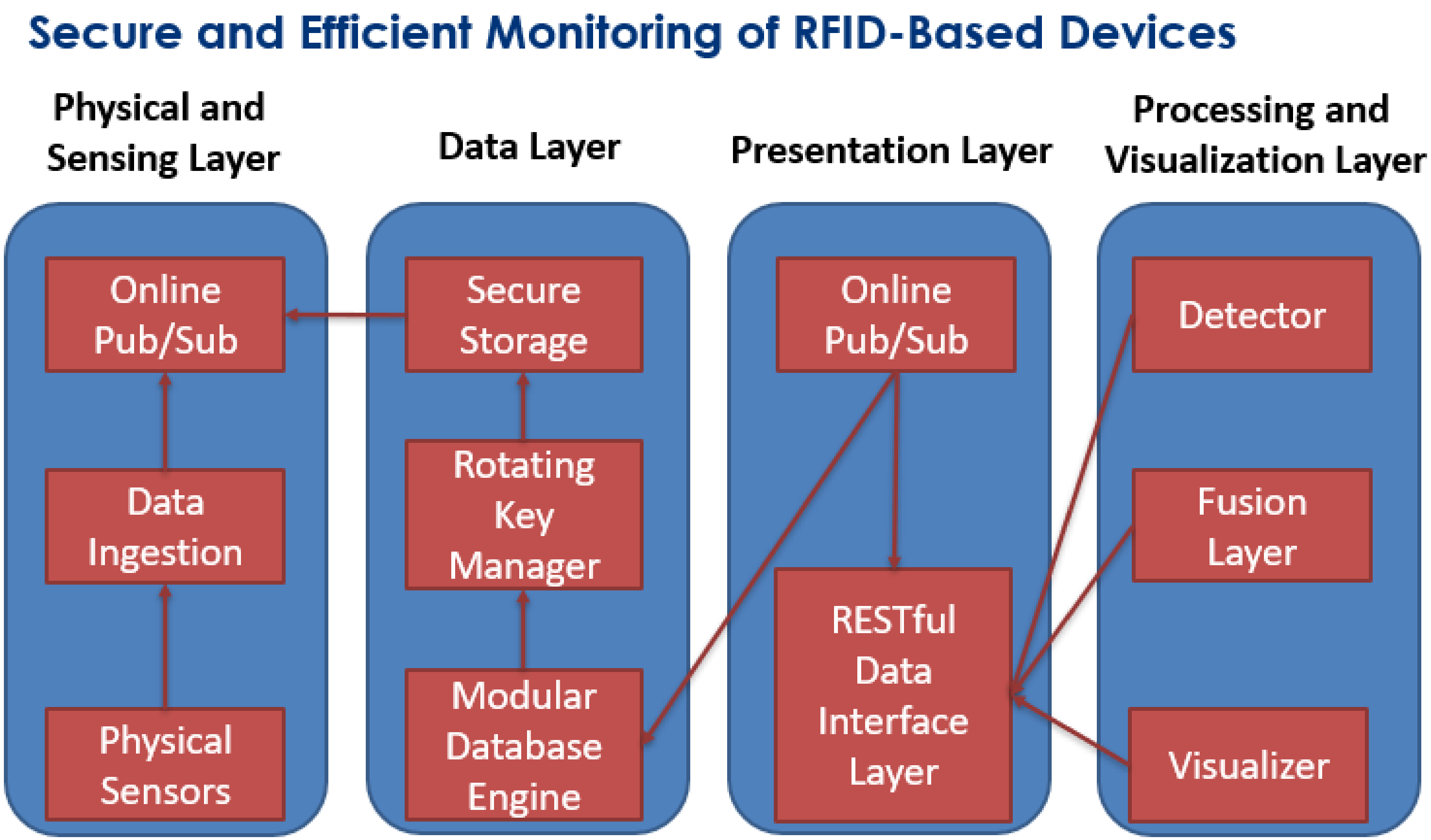
The collection framework interfaces with a heterogeneous suite of devices in real-time, and stores the data in a database or streaming service as defined by the driver configuration.

A corresponding processing suite visualizes the real-time or archived data collected by the collection framework, enabling rapid experimentation and testing of machine learning algorithms on existing and new datasets.

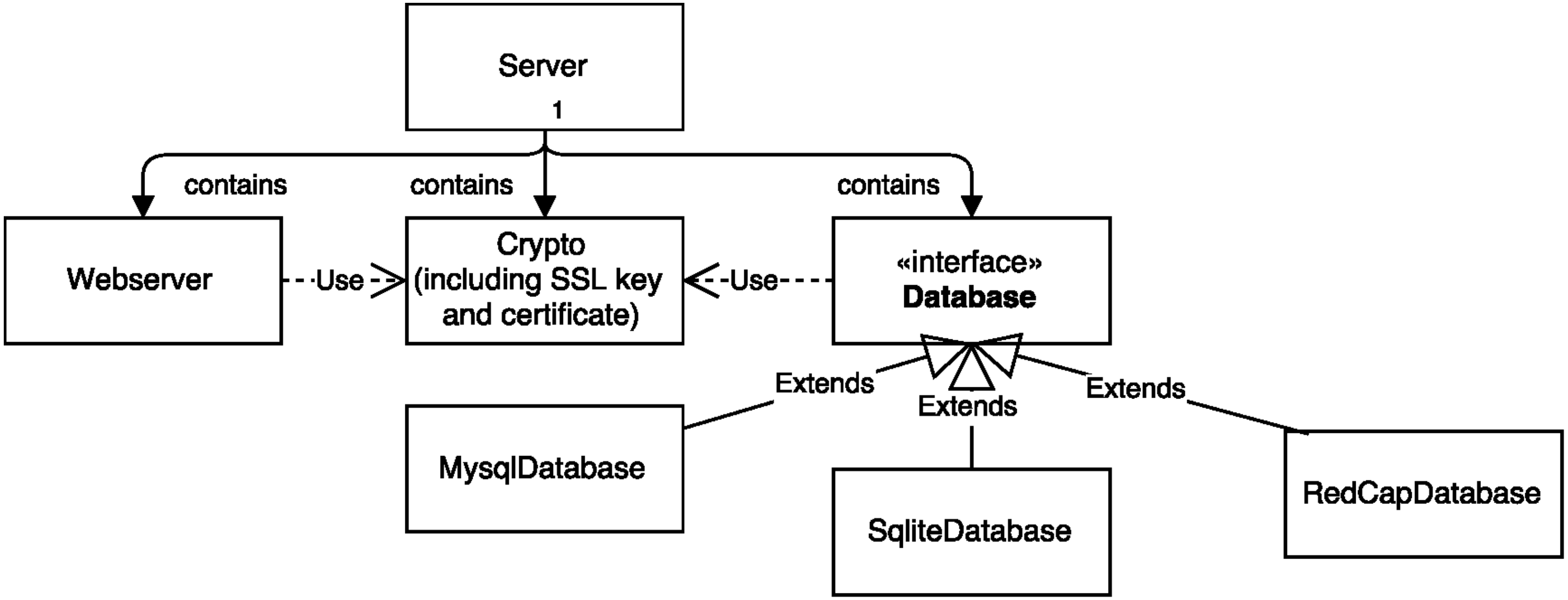
Sensor fusion, ground truth, and data perturbation modules allow for automated and controlled manipulation of the data sets and comparison to ground truth.

It is modular and generalizable to a variety of sensor systems and processing needs.

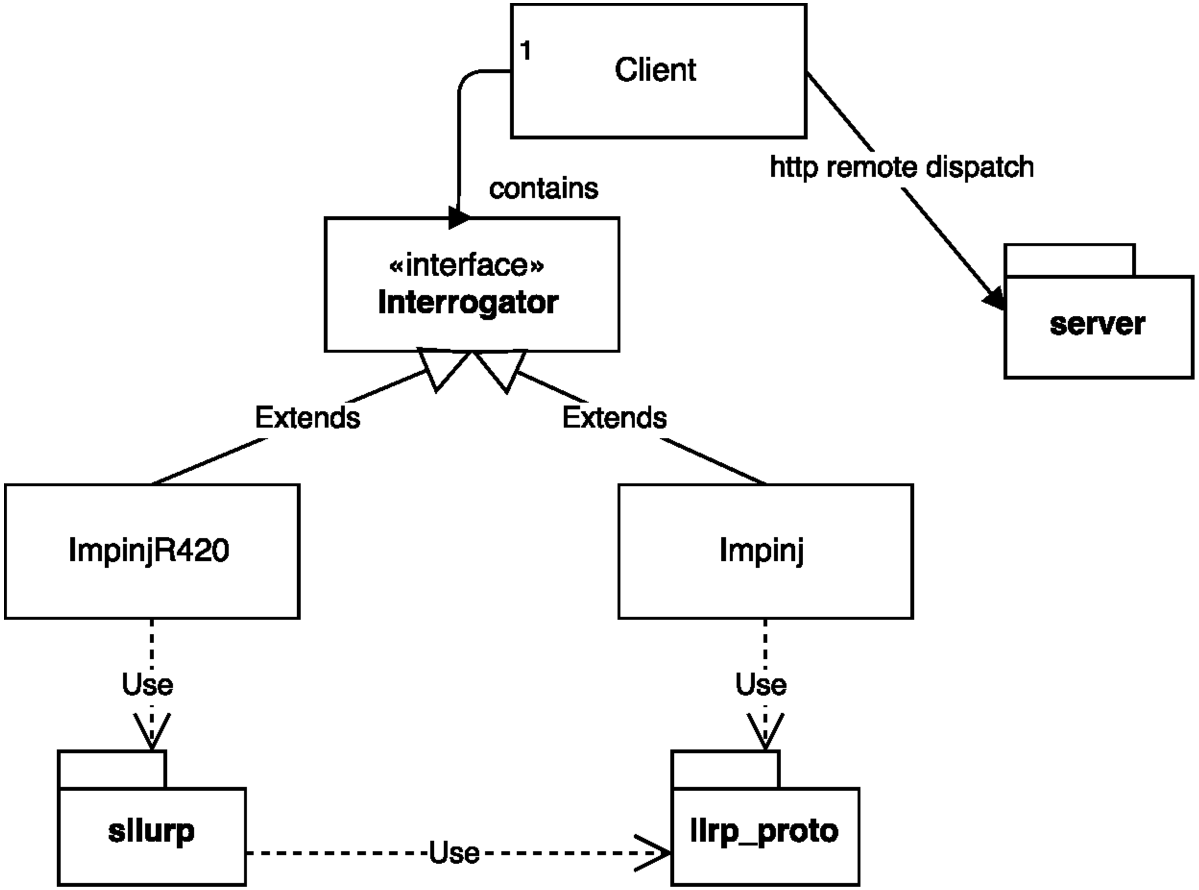
Layered Approach



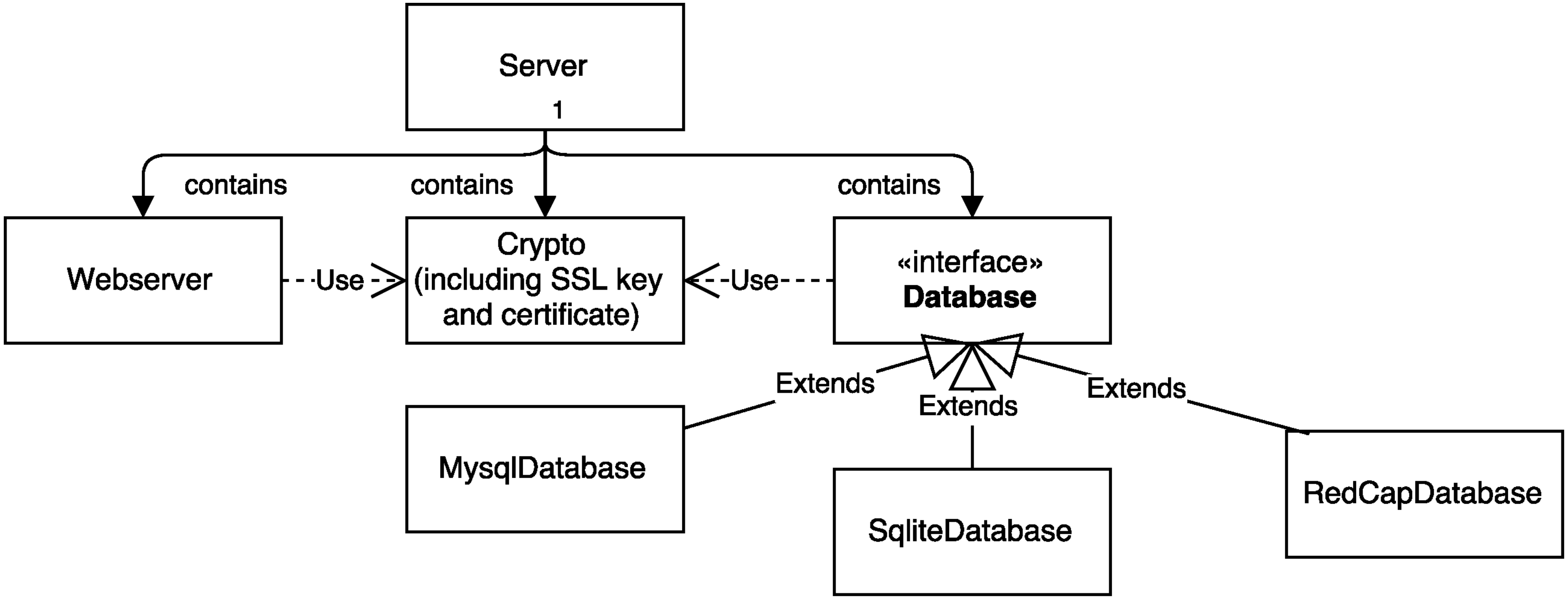
Front-End, Microservices, Back-End



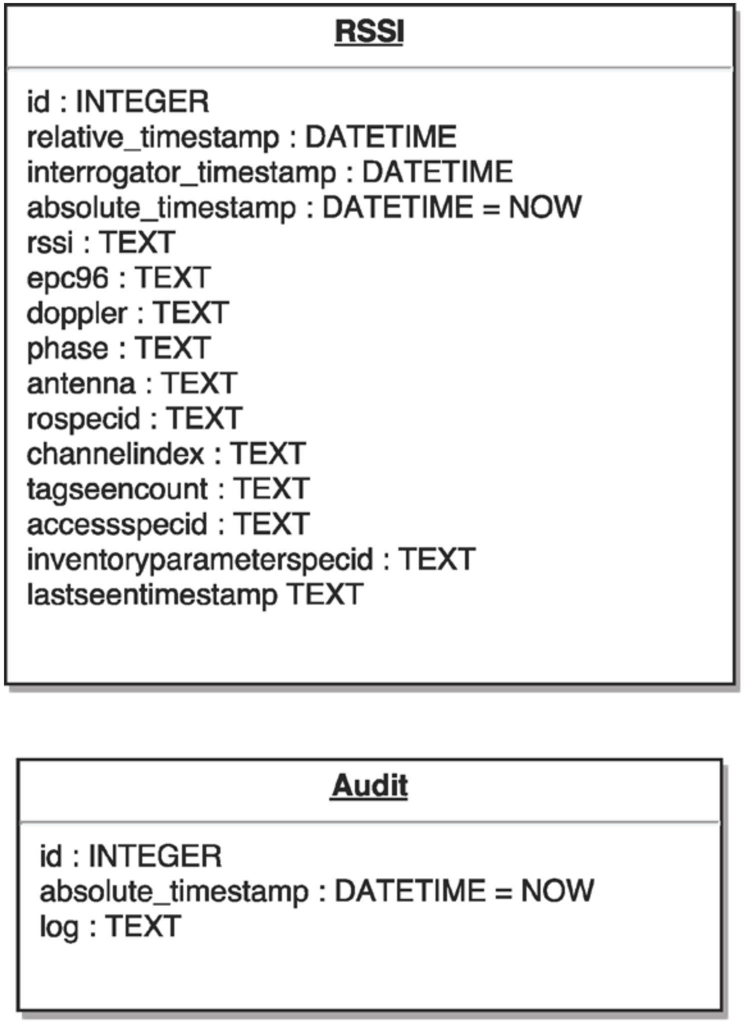
Microservice Implementation: Drivers



Microservice Implementation: Database



Database Schema



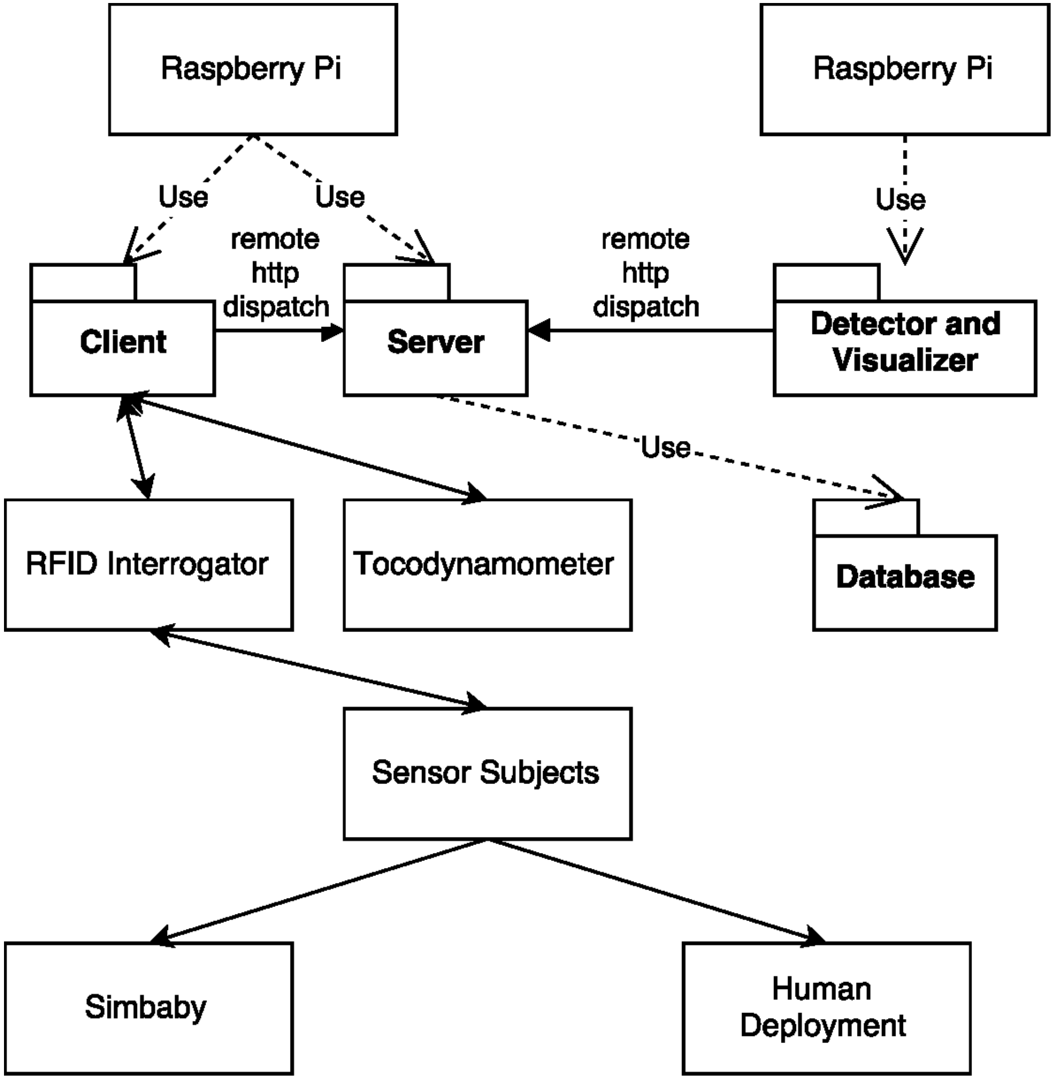
RSSI

id : INTEGER
relative_timestamp : DATETIME
interrogator_timestamp : DATETIME
absolute_timestamp : DATETIME = NOW
rssi : TEXT
epc96 : TEXT
doppler : TEXT
phase : TEXT
antenna : TEXT
rospecid : TEXT
channelindex : TEXT
tagseencount : TEXT
accessspecid : TEXT
inventoryparameterspecid : TEXT
lastseentimestamp TEXT

Audit

id : INTEGER
absolute_timestamp : DATETIME = NOW
log : TEXT

Microservice Composition



Case Study: IoT Sensor Framework

This software is available as an open source package for others to use, modify (by forking the repository and issuing pull requests), and contribute back.

- https://zenodo.org/record/3786933
- https://zenodo.org/record/3825126

Example: Reading RESTful Data

```
def retrieve_data(start, end):
    global timescale
    global db_password
    global server

    resp, content = sendhttp(server + '/api/iot/' + str(start * timescale) + '/' + str(end * timescale), headerdict={'Content-Type': 'application/json'}, bodydict={'data': {'db_password': db_password}}, method='POST')

    return resp, content
```

References

Mongan, W., Shevertalov, M., Mancoridis, S., "Re-engineering a Reverse Engineering Portal to a Distributed SOA." IEEE Proceedings of the 16th International Conference on Program Comprehension (ICPC), 2008.

Mongan, W. "A Service-Based Web Portal for Integrated Reverse Engineering and Program Comprehension."

William M. Mongan, Ilhaan Rasheed, Enioluwa Segun, Henry Dang, Victor S. Cushman, Charlie Rose Chiccarine, Kapil R. Dandekar, & Adam K. Fontecchio. (2020). drexelwireless/iot-sensor-framework: Public Release 1.0 (v1.0). Zenodo. https://doi.org/10.5281/zenodo.3786933

http://www.codeproject.com/Articles/28704/Programming-With-Exchange-Server-2007-EWS-Part-1

http://www.w3schools.com/webservices/ws_soap_example.asp