# Prudential ICM

## Technical Architecture

*Version: 4.6*

*Status: FINAL*

| | |
|---|---|
| *Project:* | *Prudential ICM* |
| *Date:* | *7/19/2022 5:13 PM* |
| *Document:* | *TA-100 Prudential Technical Architecture ICM_8.2.0.docx* |
| *Version:* | *4.6* |
| *Status:* | *FINAL* |
| *Classification:* | *Confidential* |

## Document Information

| Label | Description |
|---|---|
| **Organization** | Prudential |
| **Author** | Petre Todorov |
| **Manager** | Sander de Rooij |
| **Project** | Prudential ICM |
| **Document ID** | TA-100 |
| **Filename** | TA-100 PRUDENTIAL TECHNICAL ARCHITECTURE ICM_8.2.0.DOCX |
| **Date last updated** | 19 July 2022 |
| **Version** | 4.6 |
| **Status** | FINAL |
| **Classification** | Confidential |

## Version History

| Version | Date | Status | Comments |
|---|---|---|---|
| 0.1 | 10-03-2010 | DRAFT | Initial draft version |
| 0.2 | 15-03-2010 | DRAFT | Extended draft version |
| 0.3 | 16-03-2010 | FINAL | Incorporated internal review comments. |
| 0.4 | 16-10-2010 | FINAL | Updated with PoC information. |
| 0.6 | 27-10-2010 | FINAL | Added reference document “GHO Solvency II IT Application Architecture Design & Roadmap”<br><br>Closed open issue 6, all input will be in IDEAL format<br><br>Added sandbox and grid server environments to deployment section |

| | | | |
|---|---|---|---|
| | | | Updated infrastructure information with more details on target platform<br><br>Closed open issue 7, target platform indicators taken from “GHO Solvency II IT Application Architecture Design & Roadmap”<br><br>Updated connectivity section with evolving insights around integration with Algorithmics and MKS (caveat open issue 11 and 13)<br><br>Added issues for pending Asset and Result LDM incorporation (14 and 15) |
| 0.7 | 25-11-2010 | FINAL | Updated to include outcome of MKS Deploy / Sandbox discussion on November 11th 2010.<br><br>Closed open issue 13.<br><br>Updated Deployment Model.<br><br>Updated Connectivity description for MKS Deploy.<br><br>Also reworded 6.1.4 Asset connectivity description to reflect the temporary asset ‘datamart’ as a part of the operational db. |
| 0.8 | 28-02-2011 | FINAL | Updated to include active directory, assumption set run on GEP environment and RSG integration.<br><br>Updated deployment model and modified descriptions to include GEP.(3.1.2.1, 3.1.2.2, 3.1.3.1 and 3.1.3.2)<br><br>Updated connectivity model and added descriptions for GEP (6.1.8)<br><br>Added descriptions for active directory.(6.1.9)<br><br>Updated descriptions for RSG (6.1.5)<br><br>Updated security descriptions (8.3) |

| | | | |
|---|---|---|---|
| | | | Added open issue 16 to cover pending reference documents. |
| 0.9 | 18-05-2011 | FINAL | Updated GEP details |
| 1.0 | 26-05-2011 | FINAL | Prepared for Release D+ |
| 1.01 | 01-07-2011 | DRAFT | Add detailed section related to error handling. |
| 1.02 | 05-08-2011 | DRAFT | Expanded error handling section |
| 1.03 | 22-08-2011 | DRAFT | Added section about queuing |
| 1.4 | 19-03-2011 | DRAFT | New structure<br><br>Updated entity descriptions<br><br>Updated component descriptions<br><br>Added RSG integration<br><br>Updated Asset Data Warehouse integrationQueue ManagerLoad BalancerUpdated Activity Monitoring framework description<br><br>Added Algorithmics Job Streams<br><br>Sandboxes, Pre-Production servers<br><br>Batch Runs |
| 1.5 | 10-10-2013 | DRAFT | Updated for the release 5.3.0.0<br><br>- Removed MKS integration<br>- RSG Assembly Repository<br>- Post Run data repository<br>- Removed Compiled DLM<br>- Removed Reporting Exercises<br>- Post-run manifest file<br>- Tags |

| | | | |
|---|---|---|---|
| 1.6 | 26-01-2015 | DRAFT | - Removed GEP engine support |
| 2.0 | 01-02-2016 | DRAFT | Updated for the release 5.5.0.0 |
| 3.0 | 07.06.2019 | FINAL | Updated for the release 6.5.4.0 |
| 4.0 | 05.02.2020 | FINAL | Updated for the release 6.8.0.0 for frontend clustering |
| 4.1 | 19.08.2020 | FINAL | Updated for the release 6.9.0.0 for running multiple instances of Task Runner |
| 4.2 | 08.02.2021 | FINAL | Update for the release 7.0.0.0<br>- Stochastic runs with multiple shreds |
| 4.3 | 05.03.2021 | FINAL | Update for the release 7.0.1.0 to add default supported browser |
| 4.4 | 27.04.2021 | FINAL | Update for the release 8.0.0.0 to replace Oracle by SQL Server |
| 4.5 | 02.08.2021 | FINAL | Update for the release 8.1.0.0 to add a section for the Web Services |
| 4.6 | 01.07.2022 | FINAL | Fix return code mistake in web services |

**Status Description**

| **Status** | **Description** | **Comment** |
|---|---|---|
| DRAFT | Initial version / for review | |

| | | |
|---|---|---|
| FINAL | Delivery for final review | |
| APPROVED | Approved for sign-off | |

## Open and Closed Issues

| # | Issue | Status |
|---|---|---|
| 1 | Market data integration final solution is to be defined | Closed |
| 2 | Asset Data Warehouse integration is unspecified | Closed |
| 3 | RSG integration is unspecified | Closed |
| 4 | Working assumption that Algorithmics Security will be used for Authentication. | Closed |
| 5 | Working assumption that ARA will be used for AggregationRule evaluation is unconfirmed.<br><br>AggregationRule input format for ARA is unspecified | Closed |
| 6 | Confirmation required that 'RiskWatch' input for calculations can always be provided in IDEAL CSV format, or whether RiskWatch CSVs need to be supported as well. | Closed |
| 7 | Target hardware platform is unspecified | Closed |
| 8 | Business continuity requirements to be defined | Closed |
| 9 | Working assumption with regards to authentication:<br><br>PoC will use very simple configuration file based user definition<br><br>End State TBD | Closed |
| 10 | Lite Model vs. Assumption Set in MKS needs clarification | Closed |
| 11 | Target batch execution environment for Algorithmics and integration needs to be aligned with deployment model and connectivity overview, pending TDD updates relevant for SF phase 1 development. | Closed |
| 12 | The conceptual model needs to be re-aligned with the evolving system requirements. The new "Entity Set" concept is currently missing. It also needs | Closed |

| | | |
|---|---|---|
| | to be updated with details around the sandbox, pre-production and Algorithmics Batch run specifics when these details have been cleared. | |
| 13 | Details around MKS Deploy integration need to be defined. | Closed |
| 14 | Asset LDM integration pending. | Closed |
| 15 | Result LDM integration and feed to OBI pending. | Closed |
| 16 | Requires additional detail, referencing and alignment with final input TDD documents for:<br><br>Excellian: GEP integration (minimal impact)<br><br>RiskCare: RSG integration details (clearer definition of integration entry points required)<br><br>MKS: More detail about the MKS Source, Deploy and available version lists for GEP and RSG model integration points required. | Closed |
| 17 | Error handling related items:<br><br>How to deal with multi-line logging<br><br>Security for error message passing<br><br>Can RUNID be determined in all cases?<br><br>Is the pattern expressiveness sufficient to cover all potential logging parsing | Closed |

## Approvals

| Name | Role | Signature | Date |
|---|---|---|---|
| | | | |

## References

| Doc-id | Document | Version |
|---|---|---|
| | User Guide_Prudential_Final | |
| CF-001 | System Configuration | |

| | | |
|---|---|---|
| BP001 | Entity Structure | |
| BP002 | Lite Models | |
| BP003 | Scenarios | |
| BP005 | Assumption Set | |
| BP006 | BRP Template Configuration | |
| BP009 | User Manager | |
| BP010 | System Administration | |
| BP013 | Aggregation Rules | |
| BP014 | Entity Set | |
| BP020 | General Requirements | |
| BP022 | RAFM Projects | |
| BP023 | RSG Instructions | |

## Review Table

Please mark an **X** on the document version you have reviewed.

This will enable the Project Board approval process to be sped up.

| Document Version | 1.4 | 1.4.2 | 1.5 | 2.0 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | 4.4 |
|---|---|---|---|---|---|---|---|---|---|---|
| Paul Ross | | | | | | | | | | |
| Jean-Sacha Melon | X | X | | | | | | | | |
| Ahmed Osman | | | X | X | | | | | | |
| Deepthi Jangareddi | | | X | X | | | | | | |
| Laurent Severac | | | | | | | X | X | X | X |
| Petre Todorov | | | | | | | | X | X | X |

| Document Version | 1.4 | 1.4.2 | 1.5 | 2.0 | 3.0 | 4.0 | 4.1 | 4.2 | 4.3 | 4.4 |
|---|---|---|---|---|---|---|---|---|---|---|
| Sander de Rooij | | | | | X | | X | X | X | X |

## Confidentiality Classification

Based on the SECONDFLOOR standard confidentiality classification, this document can be classified as a **restricted** document. This means that the document will stay within a protected group. It is the readers' responsibility to handle this document as required.

## Table of Contents

# 1 Introduction

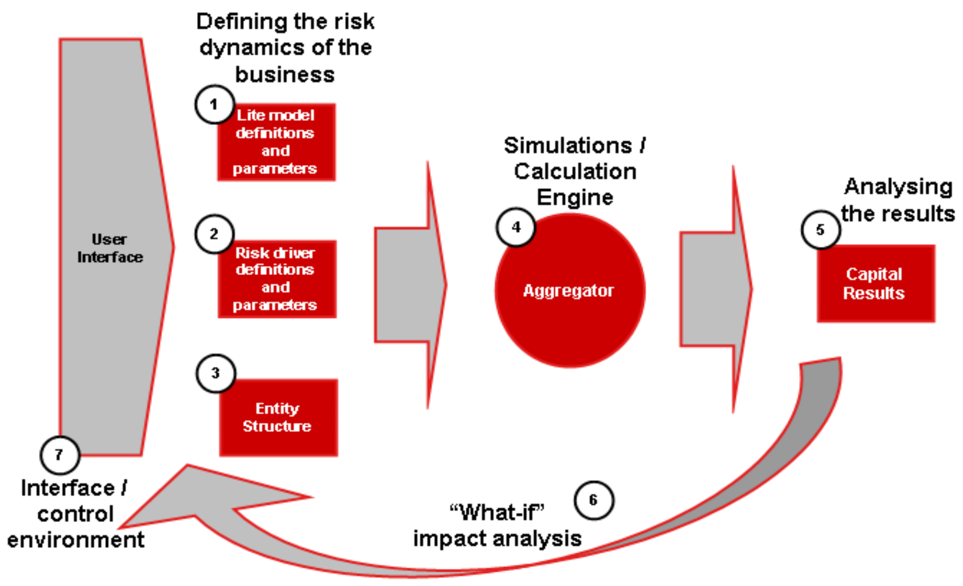
## 1.1 Background

A decision was made by Prudential (Prudential) and SecondFloor Holding BV (SF) to jointly engage in a definition study to put in place a web based solution of Prudential's Internal Capital model. Prudential has as a starting point – Capital Workflow Manager (CWM) which is a part of the Algorithmics suite to measure, manage and report Solvency Capital Requirements (SCR) under Solvency II regulations.

Prudential plc is an international financial services company with a product range including insurance, pensions and retail investments, institutional fund management and property investments.

Prudential in the UK are the leading life and pension's provider with around seven million customers. M&G was acquired by Prudential in 1999. It is the Group's UK and European fund manager. In Asia, Prudential is the leading European life insurer with life and fund management operations in 13 countries. In the US, Jackson National Life, a leading life insurance company is owned by Prudential.

The SF solution should incorporate & support all the below ICM components suggested by Prudential:


Defining the risk dynamics of the business
User Interface
1
Lite model definitions and parameters
2
Risk driver definitions and parameters
3
Entity Structure
Simulations / Calculation Engine
4
Aggregator
Analysing the results
5
Capital Results
6
"What-if" impact analysis
7
Interface / control environment

## 1.2 Scope of the document

This document describes the technical architecture of Prudential ICM: a risk management platform for Prudential to support the implementation of Solvency 2.

The content of this document covers the ICM Release 8.2.0.0.

## 1.3 Purpose of the document

The purpose of this document is

- Describe the High Level Design of the system
- Highlight key technical decisions, options and assumptions
- Inform installation & configuration at Prudential's environments
- Assist support team, by providing a reference for operations and troubleshooting
- Inform the formation of the SIT plan
- Provide requirements traceability

## 1.4 Target readership

The target readership for this document is:

- Design Teams
- Development Teams
- Test Teams
- Application Managers
- Technical Support

### 1.5 UML Notation

All UML diagrams conform to the UML specification version 2.0 as defined by the OMG (see http://www.omg.org/spec/UML/2.0).

# 2 Key Assumptions

RSG (Scenario Assumption Set runs)

- Batch server supporting the RSG calculation engine installed and configured

RAFM (Assumption Set runs & RSG Standalone runs)

- Batch server supporting the RAFM calculation engine installed and configured
- WTW Task Runner is installed and configured on each RAFM batch server

Post-processing (BU post-processing runs)

- Batch server supporting the POST_PROCESSING calculation engine installed and configured
- The fronted server must have access to the shared network location used on the batch server to store the results of BU runs.

# 3 Specific Requirements

This section informs Prudential about required infrastructure components like Java version, default supported browser, database management system.

There will be no recommendations for hosting, network or other hardware components. The need for such infrastructure parts can only be defined in relation to the intended use, e.g. the number of users in total and the numbers of users concurrently accessing the application. It is not feasible, and would also not be of much help for Prudential, to assess the need on a theoretical base.

## 3.1 Java Virtual Machine

The system runs on version 8.x (1.8) of the Java platform. The latest update of Java 8 needs to be installed on the servers.

## 3.2 Database

SQL Server 2016 shall be used.

## 3.3 Application server

Tomcat Version 8.5.x will be used. Tomcat will be delivered by SecondFloor together with the delivery of the application, including the configuration.

Deployment is carried out by Prudential using Apache Tomcat deployment process.

## 3.4 Supported browser

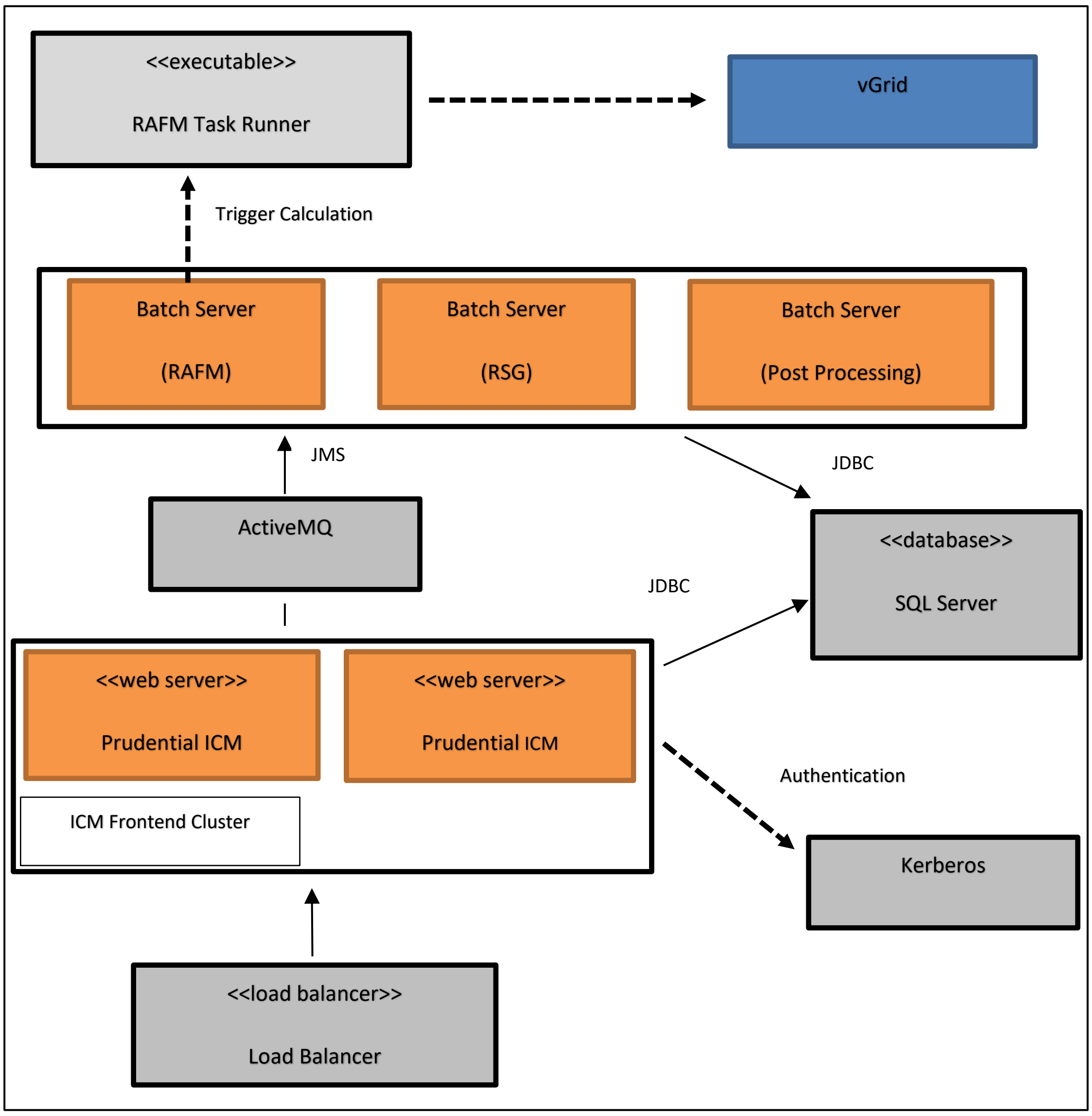
ICM supports as a default browser “Google Chrome”.

# 4 Logical System Design

This chapter provides an overall logical view of the ICM System. It is intended to describe the main components of the system as well as the high level design. The first section gives an overview of the overall system, including external components. The other sections describe each component in more details.

## 4.1 Overview

The ICM System consists of a number of components. The following diagram shows the main components and their interactions. The components developed and delivered by SF are highlighted in orange.



### 4.1.1 ICM Interface

The ICM Interface is provided by Second Floor components. It consists of an interface for the users to setup the components that compose an Assumption Set and are used to estimate Solvency II capital requirement for risk management and regulatory reporting purposes.

The ICM Interface keeps track of all changes made on its components. For every change made, an event containing the user who made the change, the timestamp of the change and what changes were made is stored. These events construct a complete audit trail.

It is also responsible for preserving the state of the components that are approved by the use of versions. When users change these approved components, the ICM Interface automatically creates a new version of that component, preserving the previous approved state.

It also controls access to the data so that it checks geographical rights of a user and provides access only to components owned or shared with the user's geography.

Besides the User Interface, ICM Interface is also responsible for the orchestration of other ICM components, by controlling their execution, generating input data needed on the components' specific formats and collecting their outputs and making them available to the users.

#### 4.1.2 ICM Calculation Engine (RAFM)

The ICM Calculation Engine is provided by WTW with RAFM. RAFM is responsible for producing the numbers for estimating Prudential's Solvency II Capital Requirements.

These numbers are produced based on the Scenario data and Market data that are applied to the Lite Models and Aggregation Rules that are configured in tree structures that represent Prudential's business structures.

The ICM calculation engine supports two types of ICM runs: Assumption Set runs and RSG Standalone runs.

RAFM Input Files Builder from the ICM Interface (see section RAFM Input Files Builder) generates the input files for the ICM Calculation based on this data. After the files are generated, the ICM Interface triggers the execution of RAFM through Task Runner.

After the Calculation is finished, ICM extracts the reports that are then available for the ICM users in the ICM Interface.

RAFM Task Runner runs asynchronously on dedicated servers and can execute multiple calculations in parallel. The number of concurrent calculations supported by a given server can be configured in the configuration file. The RAFM Task Runner execution is controlled by the Load Balancer from the ICM Interface (see section Load Balancer).

For more details on the integration between the ICM Interface and RAFM, please refer to section RAFM Integration.

### 4.1.3 RSG Instruction Set Calculation Engine (RSG)

The RSG instruction set calculation engine is an *internal* engine responsible for processing of *Scenario Assumption Set* runs. Its task is to process the data from the input translators referenced by the scenario assumptions set, write that data as a set of CSV files (the RSG Instruction Set) and store that set of files as a ZIP archive. The format of the RSG instruction set files is specified in the RSG RAFM model inputs document.

### 4.1.4 Post-Processing Calculation Engine (POST_PROCESSING)

The Post-processing calculation engine is an *internal* engine responsible for post processing of reports of a given assumption set run.

### 4.1.5 ICM Reporting

A completed Assumption Set run produces the reports and ICM stores these reports in the ICM database and makes them available to the user. For each purpose / type of Assumption Set runs there is a set of predefined reports that will be produced by RAFM and picked up by ICM. In case one of the expected reports is missing, ICM will raise an error and mark the run as failed.

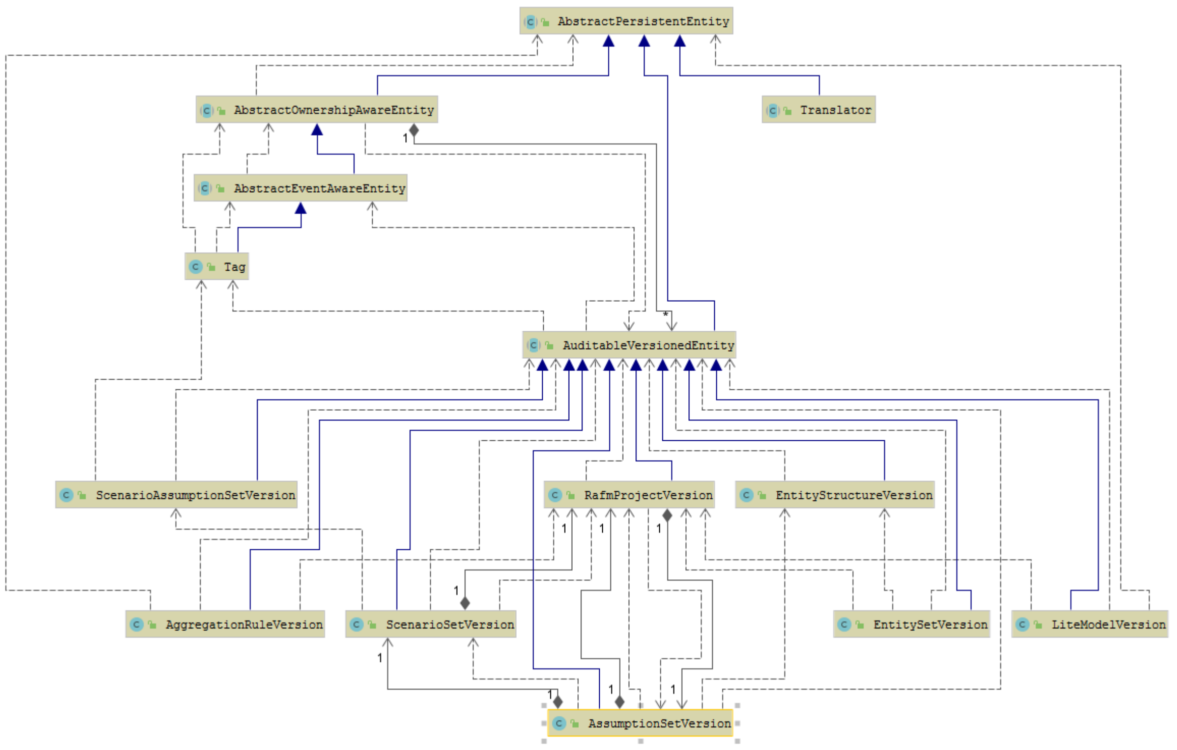
#### *4.1.5.1 Stochastic Extraction reports*

The user can also request creation of stochastic extraction reports. These reports are being stored in the ICM database like the regular reports. However, due to their size, they are made available to the users only for a given period of time. Once the predefined grace-period has expired, these reports will be automatically deleted from the ICM database.

## 4.2 System Components

This section describes the main ICM Interface components. The first section gives an overview of the main model showing the main components and how they are related to each other in a class diagram. The other sections describe each component in more details.

### 4.2.1 Overview



*AbstractPersistentEntity* is a base class for all entities in the ICM Interface that are stored in database with its own identifier – a unique number generated from database sequence and assigned to the entity automatically when it is saved to the database.

Some entities in the ICM Interface can have multiple versions of the same “parent” entity. *AbstractOwnershipAwareEntity* and *AbstractEventAwareEntity* are base classes for all “parent” entities.

*AbstractOwnershipAwareEntity* encapsulates such information as name of the entity, a user who created the entity and a user group that owns the entity as well as user groups the entity is shared with. This allows controlling the access to the entity.

*AbstractEventAwareEntity* holds collection of events that allows keeping track of all changes made on the entity.

*VersionedEntity* and *AuditableVersionedEntity* are base classes for all versions of ICM components such as Entity Structure Version, Lite Model Version, etc.

*VersionedEntity* has a parent and a version number and *AuditableVersionedEntity* holds a status of the entity. When an entity is created it automatically gets the status “In Review”. After the entity is reviewed by a user, it gets one of the final statuses such as Validated or Rejected. Please refer to the BP documentation for more details about the possible statuses.

The *Version number* is used to track changes of an entity over time. When an entity is created it is automatically assigned the version 1.0. A new version of the entity is created each time when Validated or Rejected entity is modified so that the previous state is preserved. When an entity is deleted all its versions are deleted as well.

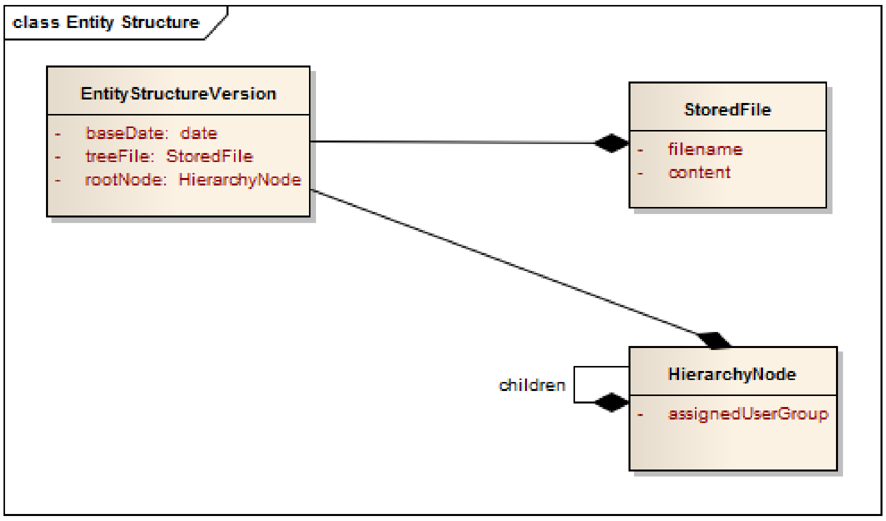
### 4.2.2 Entity Structure

Entity structures represent the business structure and contain the business nodes arranged in a parent-child hierarchical relationship. This tree structure defines the relationship between the nodes in the overall structure as well as their position in the geographical structure.

Entity structure nodes may have a Lite Model or an Aggregation Rule attached, depending on their position in the hierarchy. Lite Models are assigned to leaf nodes while Aggregation Rules are attached to parent nodes in the hierarchy.

This structure can define the whole Prudential's Business Structure as well as only a slice of it. This allows a great flexibility to manage different Business Units as well as to test different scenarios where this structure could be changed for a particular case.

The following class diagram shows the entities that support Entity Structures in ICM.



Please refer to BP001 document for a detailed definition of the functionality available for Entity Structures.

### 4.2.3 RAFM Project

RAFM Projects are artefacts developed within the RAFM application (external to the ICM interface) but tracked in ICM. The RAFM application allows a user to write modelling code, set parameters, define model runs with the aim of producing specific output that can be saved locally on the users drive or server.

The RAFM Project once produced from the RAFM application is a ZIP archive made up of a number of files that are to be stored and versioned in the ICM Interface as a new 'RAFM Project'.

The RAFM Project contains multiple files that are necessary for running an assumption set calculation either through Task Runner or manually in RAFM. It also contains a set of CPP files that are known as "submodels" and they contain the code for the API functions, the RSG implementation, the Lite Model and Aggregation Rule code and the Bridge Aggregator submodels.

There are three types of RAFM projects:

- Base Engine
- Standard Project
- Merged Project

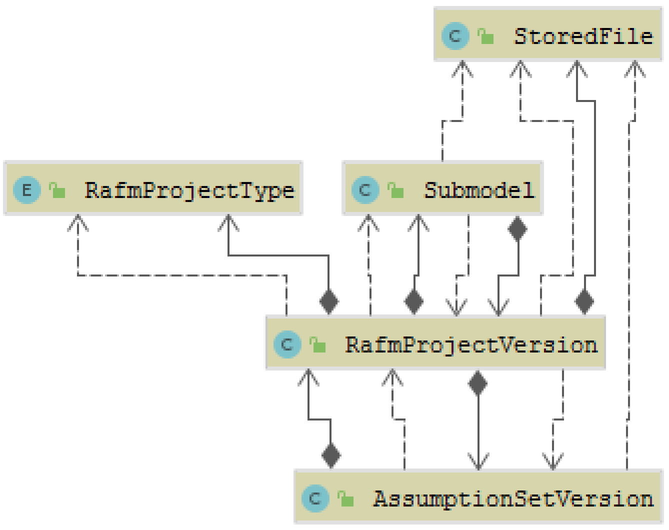
A **Base Engine** is a RAFM Project that contains the API and the RSG submodels. These submodels are grouping of multiple CPP submodel files and are threaded as a single submodel.

A **Standard Project** is a RAFM Project that is built upon a specified Base Engine but it also contains Lite Model and Aggregation Rule submodels. These submodels contain Lite Model code / Aggregation Rule code that can be associated with a given Lite Model or Aggregation Rule within the ICM user interface.

A Standard Project is created by selecting a Base Engine and uploading a RAFM Project archive. During the upload the ICM user interface will perform a consistency check to ensure that the uploaded ICM RAFM Project is consistent with the selected Base Engine.

A **Merged Project** is a type of RAFM project that can be assigned to an Assumption Set that has nested entity sets. It is based on a selected Base Engine and a number of Standard Projects. The Merged Project is created manually in the RAFM tool and it consists of the specified base engine, a union of all Lite Models and Aggregation Rules from all included Standard Projects and the relevant Bridge Aggregator submodel from each Standard Project.

A Merged Project is created by selecting a Base Engine, Assumption Set with nested entity sets and uploading a RAFM Project archive. During the upload the ICM user interface will perform a consistency check to ensure that the uploaded ICM RAFM Project is consistent with the selected Base Engine and the Standard Projects references by the main entity set and the nested entity sets.



*4-1 RAFM Project class diagram*

### 4.2.4 Lite Model

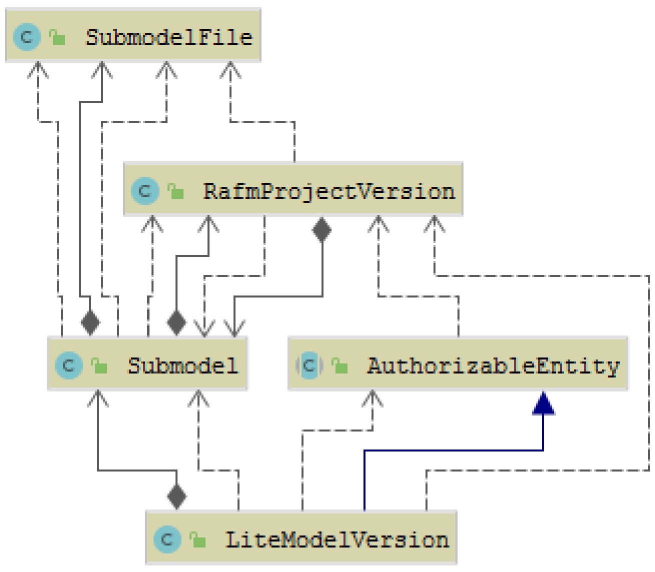
Lite Models are a combination of inputs that allow the calculation of assets and liabilities of an entity in the Prudential corporate structure under any given scenario. An instrument will represent the values expressed by this model by capturing the model calculation results in its attributes.

A Lite Model consists of a General Parameters file, Experience Parameter file and a reference to a Standard RAFM project and a submodel of type Lite Model that contains the Lite Model code associated with the Lite Model.

The General Parameters and Experience Parameter files are Excel files uploaded in ICM interface that provide static values used in a calculation.

For every Lite Model the ICM interface keeps the history of all previously uploaded files (code, general and experience parameters files) so that during modification the users can select one of the previously uploaded files as well as upload a new one.

Lite Models can be validated based on 4 eyes principle after being used in a successful run.



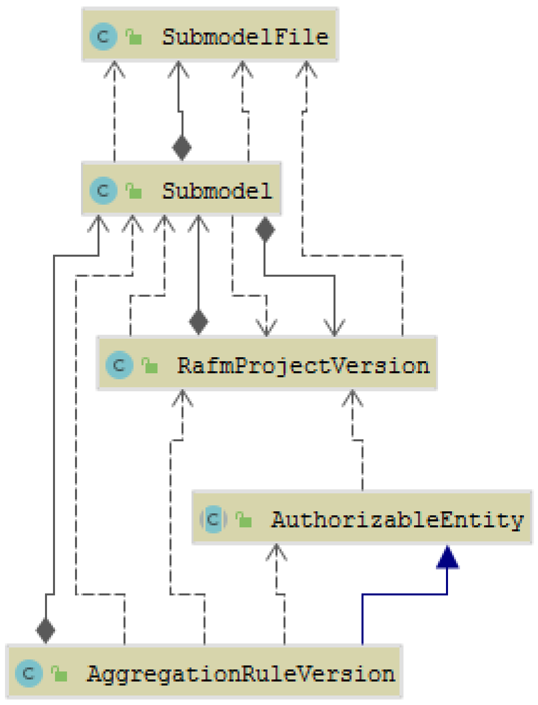
*4-2 Lite Model class diagram*

### 4.2.5 Aggregation Rule

Aggregation Rules are responsible to combine lower level results and make them available for upper level nodes. These Aggregation Rules are attached to non-leaf nodes in the Entity Structure. Aggregation Rules consist of a Parameter file and a reference to a Standard RAFM Project and a submodel of type Aggregation Rul.

The code for the Aggregation Rule is written in RAFM as C++ code. However for convenience it can be viewed in the ICM interface as well.

Aggregation Rules can be validated based on 4 eyes principle after being used in a successful run.

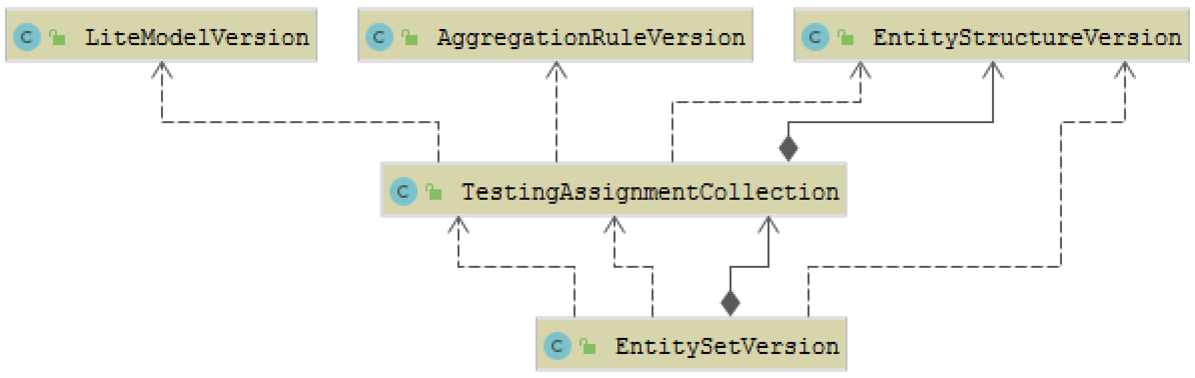


*4-3 Aggregation Rule class diagram*

### 4.2.6 Entity Set

Entity Sets are used to define how the Aggregation Rules and Lite Models are setup in an Entity Structure for an Assumption Set. Entity Sets allow users to reuse the same configuration of Lite Models and Aggregation Rules in different Assumption Sets.

An Entity Set consists of an Entity Structure and the associations of Aggregation Rules and Lite Models to each node of the structure.

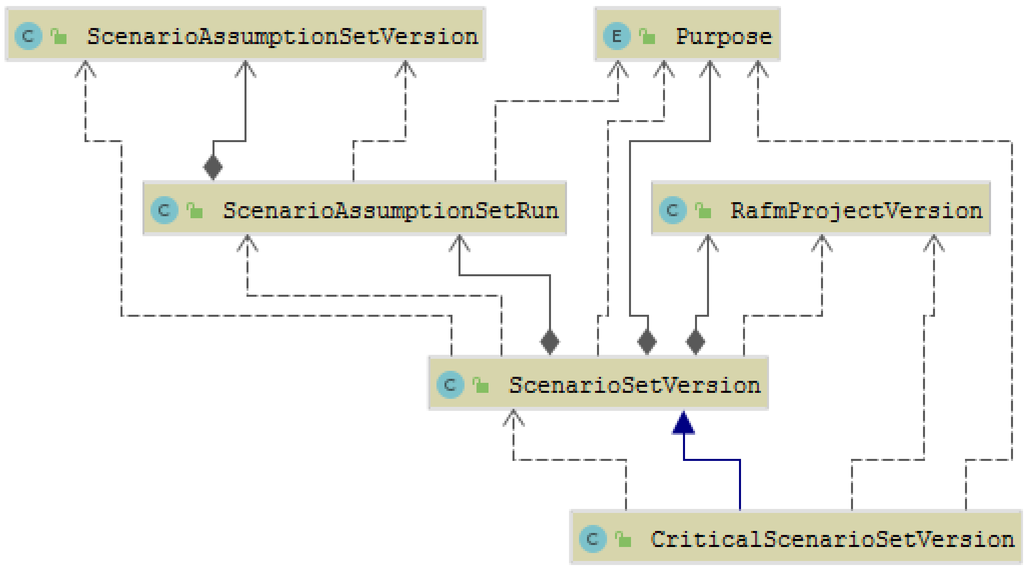


*4-4 Entity Set class diagram*

### 4.2.7 Scenario Set

Scenario sets define values for risk factors. Scenarios can be divided into *Normal* and *Critical* Scenarios. Normal Scenarios are defined by selecting a RAFM Project of type Base Engine and a successful Scenario Assumption Set run.

Critical Scenarios are based on previously executed Assumption Set runs and are created automatically after a successful Batch Stochastic & CS run.



*4-5 Scenario Set class diagram*

### 4.2.8 Input Translator

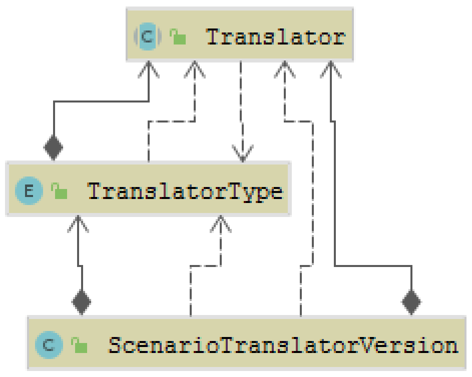
Input translators are Excel workbooks which have been developed by Capital Management & Modelling to upload the required data into IMS and starting from ICM 5.6.x directly to ICM.

There are multiple types of translators:

- Risk Driver Universe
- Dependency Calibration
- Risk Calibration
- Shreds

- Initial Values
- Specified Stress
- What-if Stretch
- What-if Value
- Validation Scenario

The data from the translators is parsed during the upload and if there are no validation errors it is stored in the system. The translator can then be used as part of a Scenario Assumption Set.



*4-6 Translator class diagram*

#### 4.2.9 Scenario Assumption Set

Scenario Assumption Sets group a static data and a number of translators together, depending on the type of the scenario set.

For every scenario assumption set the user can trigger a run that will process the information from the translator files references by the scenario assumption set, perform all validations against the static data and convert the information in the csv format thus creating an RSG Instruction set file.

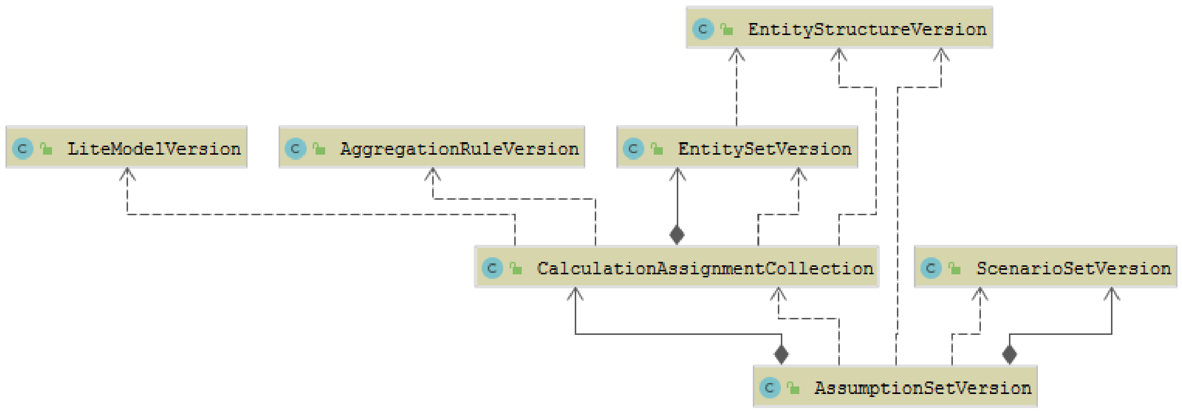
#### 4.2.10 Assumption Set

Assumption sets link all the other components together allowing users to execute their calculations by using the RAFM Calculation Engine.

The next sections describe in more details these components.

Assumption Sets are used to run the actual execution of the models. In order to define an Assumption Set, users have to previously define and setup an Entity Set setting the Aggregation Rules and Lite Models in an Entity Structure.

Using Assumption Sets users can create layers of the business by allowing lower branches to nest their Entity Sets into a higher Assumption Sets. These nested Entity Sets then become part of the higher Assumption Set, making their models part of the higher Assumption Set executions.



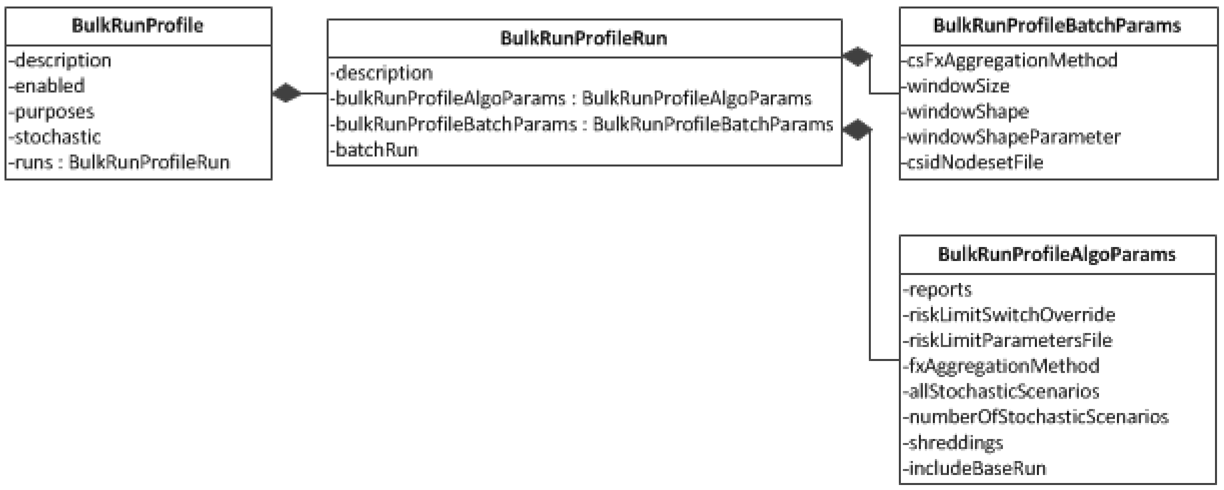
*4-7 Assumption Set class diagram*

### 4.2.11 Bulk Run Profiles

Bulk run profiles provide a way to capture a set of pre-defined run-time settings under a locked down & signed off bulk run profile.

Each bulk run profile can have multiple runs which are run templates that contain pre-defined run-time settings.

If the profile is enabled and shared with a given geography, the user can choose and run the bulk run profile. This operation will create and run all the runs defined in the given bulk run profile. The user can also choose to run a set of runs from the template or to override certain run-time settings from the profile.



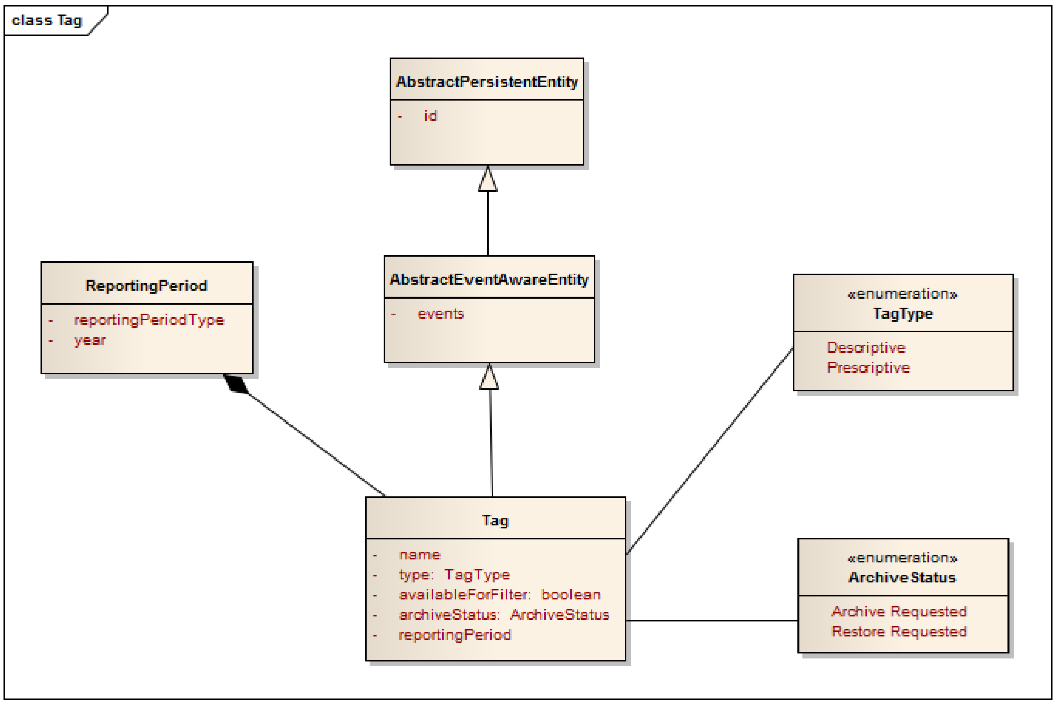
For details please refer to ***BP006 BRP Template Configuration*** and ***BP005 Assumption Set***.

### 4.2.12 Tags

Tags are used to mark the ICM components that are to be used for a specific purpose or reporting period. Tagging system in the ICM interface allows users to tag the ICM components and then to filter on specific tags that they are working on over a given period. Apart from that tags can be used to mark the ICM components for archiving in order to prepare the system for physical archiving in the future.

The ICM Interface is pre-configured with the ‘New’, ‘Rejected’ and ‘Historical’ tags that cannot be modified or deleted from the ICM. The system tags as well as user-defined tags are stored in the ICM database.

Tags can be assigned on assumption sets, entity sets and scenario sets by the users and then automatically propagated by the system to the underlying components – lite models, aggregation rules, entity structures, asset portfolios – through a set of system-driven assignment business rules.



### 4.2.13 Task Runner Configurations

Task Runner Configurations are part of the System Administration. They allow the administrator to configure into ICM the installed RAFM Task Runner tools.

Their configurations consist of a version name, the path to the RAFM Task Runner tool and a parameters template file. The parameters template file is the template which will be used to generate the Task Runner input file `RAFM_Parameters.rtk`.

### 4.2.14 RAFM Configurations

RAFM Configurations are part of the System Administration. They allow the administrator to associate RAFM Project versions to Task Runner tools using a Task Runner Configuration. By doing so, the administrator creates a mapping between a certain RAFM Project version and an installed RAFM Task Runner tool allowing multiple RAFM Project versions to be supported by ICM.

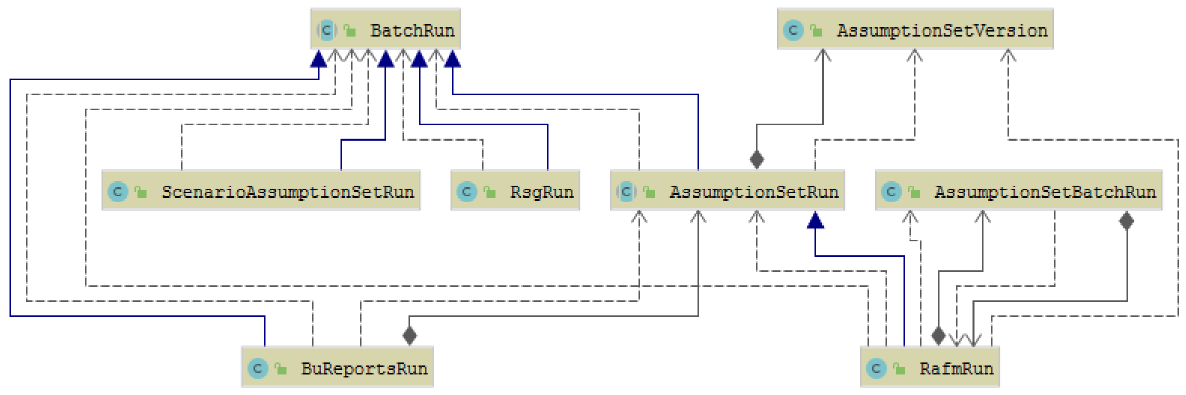
### 4.2.15 Batch Runs

Batch Runs define an asynchronous execution of internal or external components that ICM Interface integrates with. A batch run holds the information that is displayed to the users on the Run Tables on Scenario Assumption Set, Scenario Set and Assumption Set screens.

The identifiers of Batch Runs, known as Run IDs, are used to identify an execution throughout the ICM system, including external systems such as RAFM.

Separate Batch Servers are used to process different types of runs. Currently, there are four types of batch runs: RSG Standalone Runs, Assumption Set Runs, Scenario Assumption Set Runs and Post-Processing Runs. Each batch run is executed by a batch server for the calculation engine required by that specific run type (RAFM, RSG, POST_PROCESSING). Depending on the required calculation engine a suitable batch server is allocated by the Load Balancer to process the run. See Load Balancer for more details on how the Load Balancer works.

Please, refer to the BP Documents BP005 Assumption Set, BP003 Scenarios and BP010 System Administration for more details.



### 4.2.16 Batch Servers

Batch servers are the execution environments used to execute big Batch Runs. Batch Server support concurrent execution of multiple batch runs. The number of concurrent runs supported by a given server can be configured in the batch server property file.

Each Batch Server supports one Calculation Engine. This calculation engine is configured for each server with one of the available calculation engines: RAFM, RSG or POST_PROCESSING.

Load Balancer is responsible for finding a suitable Batch Server for a given Batch Run. See section Load Balancer for more details.

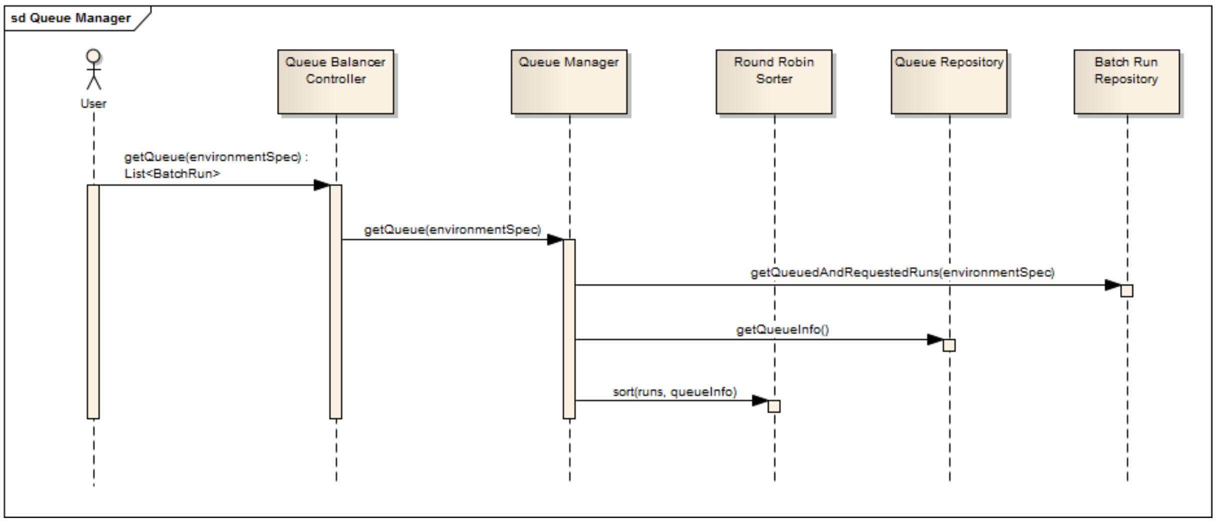
Batch Servers are managed by a System Administrator through the Status page.

### 4.2.17 Queue Manager

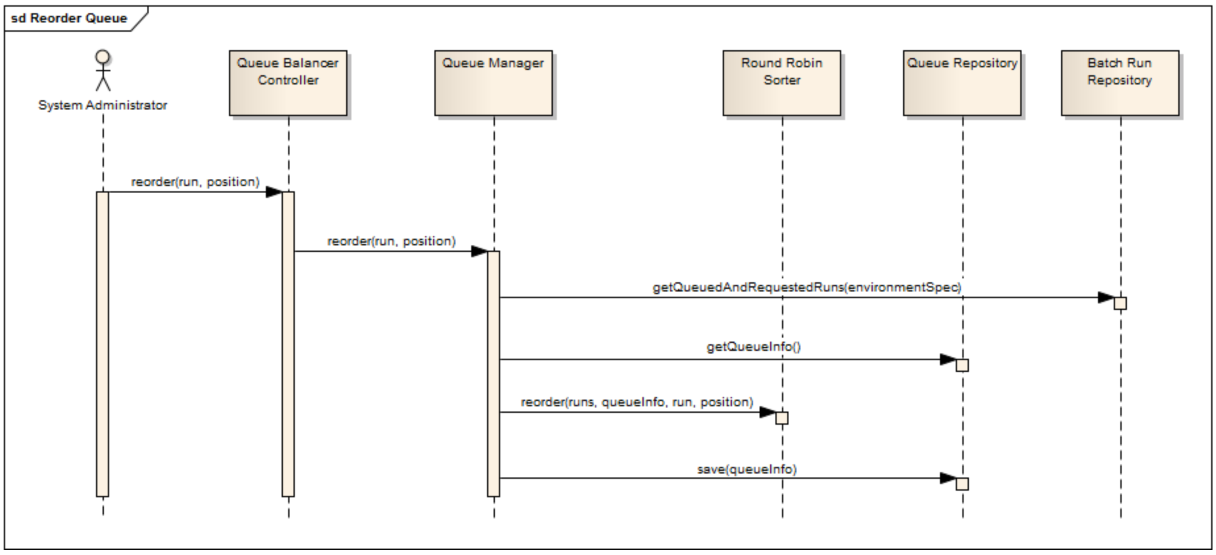
The Queue Manager is responsible for maintaining a sorted queue of batch runs available for execution. The runs are distributed to the available computing resources i.e. batch servers and

RAFM vGrid pools. As soon as there are no more available resources in the ICM System the remaining runs will be queued waiting for their turn to process. The queued runs are available on the ICM interface so that the users can see the ordered queue of runs and predict when their runs can be executed.

The Queue Manager maintains a separate queue for each calculation engine (RAFM, RSG, POST-PROCESSING). These queues are then sorted using Round-Robin sorting algorithm based on the time the run was triggered and the Business Unit from the user who triggered the run.


sd Queue Manager
User
Queue Balancer Controller
Queue Manager
Round Robin Sorter
Queue Repository
Batch Run Repository
getQueue(environmentSpec) : List<BatchRun>
getQueue(environmentSpec)
getQueuedAndRequestedRuns(environmentSpec)
getQueueInfo()
sort(runs, queueInfo)

The System Administrators can influence the sorting algorithm by manual reordering of runs. If the System Administrator manually changes the position of the run in the queue then the sorting algorithm will no longer affect the part of the queue that was reordered manually.



##### *4.2.17.1 Round-Robin Algorithm*

The Round-Robin algorithm as used for sorting runs works as follows:

First, all runs are split into separate queues per business units. Each of these virtual queues is ordered by the age or a run where oldest runs come first.

Then, to ensure fair allocation of resources over various business units, the virtual queues are combined into one queue using the Round-Robin algorithm. The Round-Robin algorithm simply iterates over all business units in a fixed order, picking the highest priority run from the current business unit and adds it to the queue before moving on the next.

##### *4.2.17.2 Example*

Let’s assume we have business units BU1, BU2 and BU3 and runs triggered from these business units in the following order:

| |
|---|
| **Run1 (BU1)** |
| Run2 (BU1) |
| Run3 (BU2) |
| Run4 (BU1) |
| Run5 (BU2) |
| Run6 (BU3) |

The Queue Manager splits these runs by business unit into virtual queues effectively ordering the runs in each queue by their age:

| | | |
|---|---|---|
| **Run1 (BU1)** | **Run2 (BU1)** | **Run4 (BU1)** |

| | |
|---|---|
| **Run3 (BU2)** | **Run5 (BU2)** |

| |
|---|
| **Run6 (BU3)** |

The Round-Robin algorithm will go over the queues in order BU1-BU2-BU3, producing the following queue:

| |
|---|
| **Run1 (BU1)** |
| Run3 (BU2) |
| Run6 (BU3) |
| Run2 (BU1) |
| Run5 (BU2) |
| Run4 (BU1) |

The Round-Robin algorithmic prevents starvation: No business unit can flood the ICM Interface and prevent other BUs from executing their runs while at the same time no BU can be left behind other BUs constantly gaining a higher priority.

Finally, the ICM application allows manual reordering. An administrator may manually alter the order of the queue.

For example, suppose that in the previous example, run6 (BU3) was moved from the 3rd position to the head of the queue. We get the resulting queue, where the first three items are now fixed:

| |
|---|
| **Run6 (BU3)** |
| Run1 (BU1) |
| Run3 (BU2) |
| Run2 (BU1) |
| Run5 (BU2) |

| |
|---|
| Run4 (BU1) |

Moreover the key order is adapted in accordance to already sorted items so that it will be BU3-BU1-BU2 instead of initial BU1-BU2-BU3.

Now if a new run - run7 - is triggered from BU3 it will get priority over runs from BU1 and BU2. The resulting queue will look like:

| |
|---|
| **Run6 (BU3)** |
| Run1 (BU1) |
| Run3 (BU2) |
| Run7 (BU3) |
| Run2 (BU1) |
| Run5 (BU2) |
| Run4 (BU1) |

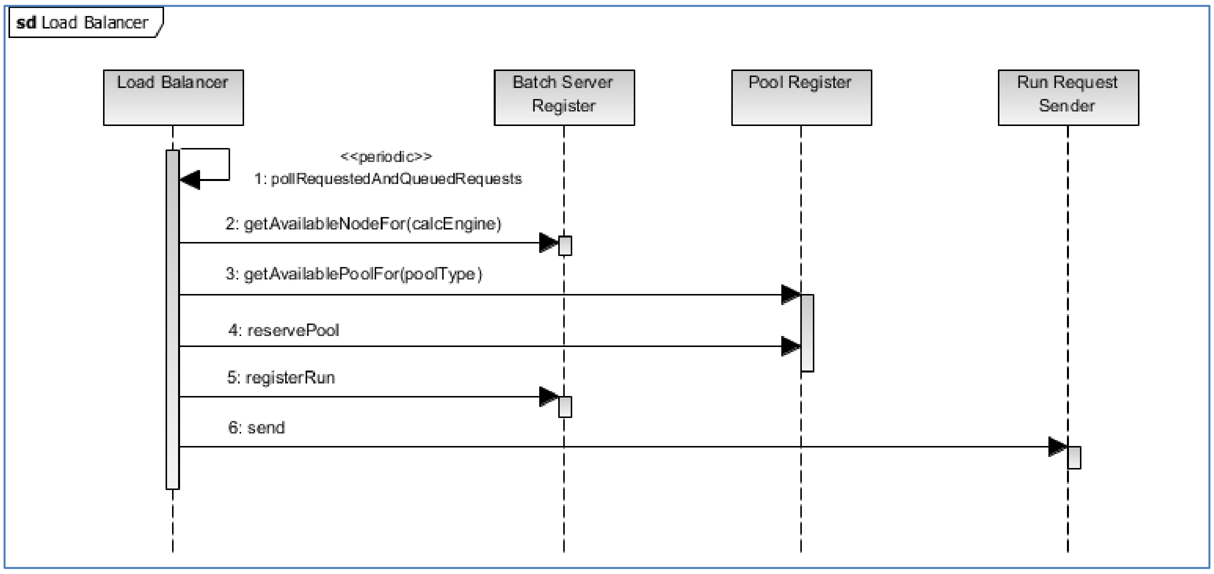
### 4.2.18 Load Balancer

The Load Balancer is responsible for controlling the execution resources: Batch Servers and RAFM vGrid Pools. The Load Balancer interacts with the Batch Server Register and the Pool Register so that it knows which resources (batch servers / pools) exist in the system, as well as controls which are available, reserved or disabled.

The Load Balancer runs in a batch process that is periodically executed. Every time it is executed, it goes through the list of all queued batch runs waiting for execution (sorted by the algorithm described in the previous section) and for each run it tries to find a suitable batch server (and in case of RAFM runs a suitable vGrid pool) and then asks the Queue Manager to execute that run on the specified batch server.

If there is a Batch Run to be executed, the Load Balancer reserves the selected vGrid pool (applicable for RAFM runs only) and then sends a message to the batch server, so that it can start executing the Batch Run. Once the Run execution is finalized, the batch server sends a message back to the Load

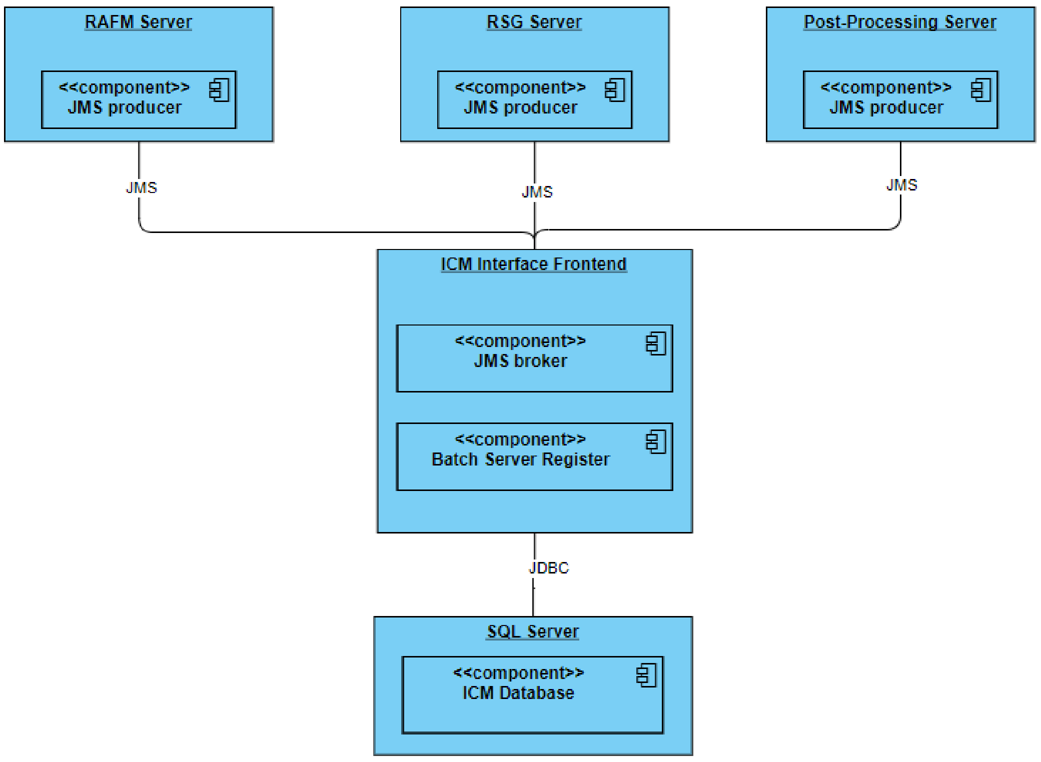
Balancer which will then mark the run as COMPLETED/FAILED and release the vGrid pool reservation so that it becomes available for a next execution.



#### *4.2.18.1 Batch Server Register*

Batch Server Register is responsible for collecting and maintaining information about all batch servers available in the ICM System. The Batch Server Register knows how many batch servers exist in the system as well as their statuses and usage details. This information is then used by Load Balancer to find suitable resources for batch runs.

Batch servers are registered in the ICM application using Heartbeat mechanism. Every batch server sends periodic heartbeat messages in some configured interval and the ICM frontend server receives and processes these messages. The communication between Batch Servers and ICM frontend is done using JMS.


RAFM Server
<<component>> JMS producer
RSG Server
<<component>> JMS producer
Post-Processing Server
<<component>> JMS producer
JMS
JMS
JMS
ICM Interface Frontend
<<component>> JMS broker
<<component>> Batch Server Register
JDBC
SQL Server
<<component>> ICM Database

A heartbeat message contains all necessary information about batch server:

- Name – batch server node name
- Calculation Engine – RAFM, RSG or POST_PROCESSING
- Maximum number of concurrent runs
- List of IDs of runs currently being processed on the server

The first three attributes - Name, Calculation Engine and Maximum number of concurrent runs - are configured in the batch server itself and cannot be changed until the batch server is restarted. In contrast the List of IDs of runs being processed is changed every time a batch server receives a run for processing or when the processing is finished.

When the ICM frontend server receives a heartbeat from a batch server it checks if the batch server with the same name is already registered. If not then it is added to the list of already registered batch servers, otherwise the information about it is updated.

The ICM application keeps the list of registered batch servers in the java process memory while reservation details are stored in ICM database. It means that as soon as the ICM frontend server is stopped this information will be lost and then after restarting it all the batch servers will be registered again with reservation information picked up from database.

#### *4.2.18.2 Pool Register*

The pool register is responsible for maintaining a list of vGrid pools available for execution of RAFM runs. ICM tracks the availability of a given vGrid pool i.e. whether it is enabled and whether it is available or in use by any run.

In case of a queued RAFM run the Load Balancer will first try to find an available batch server for execution of the run, then it tries to find a pool of the required type. ICM will then reserve the pool and run Task Runner. Once the run is complete the reservation of the pool will be released and the pool will become available for another run.

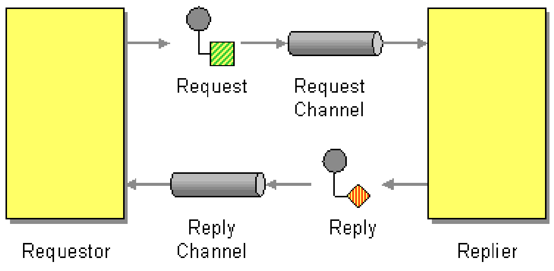
#### *4.2.18.3 Housekeeping*

Housekeeping is a process that is periodically executed to keep track of all registered batch servers. It simply iterates over the list of registered batch servers checking their statuses and cleaning reservation if needed.

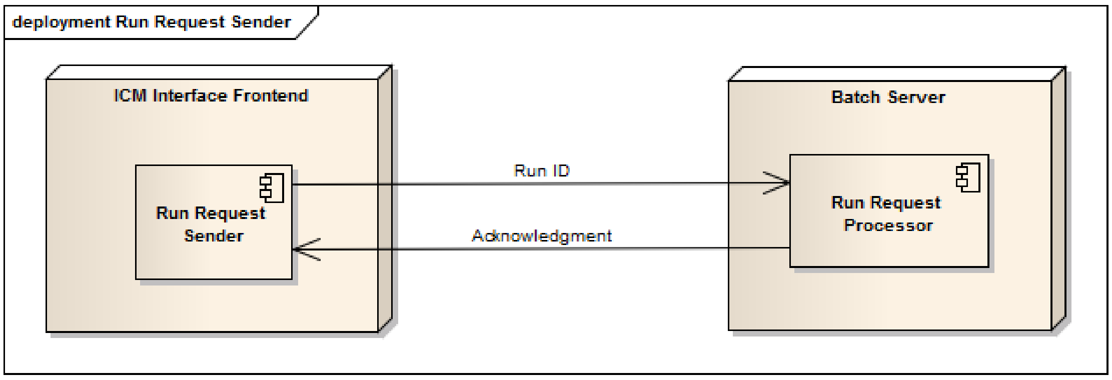
Housekeeping process is also responsible for revealing “lost” batch server nodes. If the frontend stops received heartbeats from a batch server during some configured period then this batch server is considered as lost and therefore cannot be selected by the Load Balancer for batch runs processing.

#### *4.2.18.4 Run Request Sender*

Run Request Sender is a JMS component to send Batch Runs to Sandboxes and Pre-Production servers. Request-Reply pattern is used for communication between the ICM frontend and the Batch Servers.



Run Request Sender sends a request containing run ID to the batch server reserved for that run and waits for a response from that batch server. As soon as the request reaches the destination an acknowledgement message is immediately returned to the ICM frontend to indicate that the batch server picked up the request and started its processing.



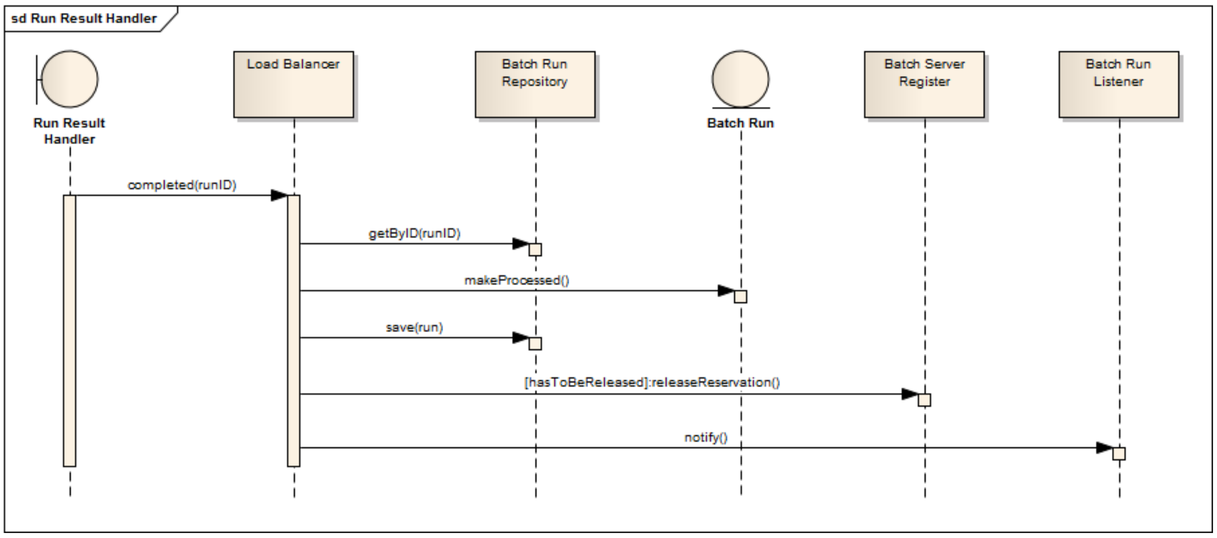
If the acknowledgement is not received during some configured timeout or any other exchange problems occur then the run will be queued again so that it can be picked up by the Load Balancer and sent to another available batch server.

In case if the acknowledgment is received but it’s not successful then the run will be immediately failed in the ICM interface with the failure message taken from the acknowledgment message.

#### *4.2.18.5 Run Result Handler*

Run Result Handler is a part of the Load Balancer that is responsible for handling successful or failed run results.

Run execution can take a long time on a batch server. When it is finished the batch server sends a message using JMS to the ICM application indicating that the execution is completed or failed. If it’s failed then the batch server also sends failure details. The Run Result Handler changes the run status in ICM database and releases the reservation of the vGrid pool if necessary. It also notifies Batch Run listeners, if there are any, that the run was completed or failed.



### 4.2.19 Mail Notifications

Mail notifications are sent after a completed (successful or failed) run. The component responsible for sending mail notifications is a scheduled task that periodically checks whether there are completed runs that have no notification sent yet and then it creates and sends a mail notification for each one of them.

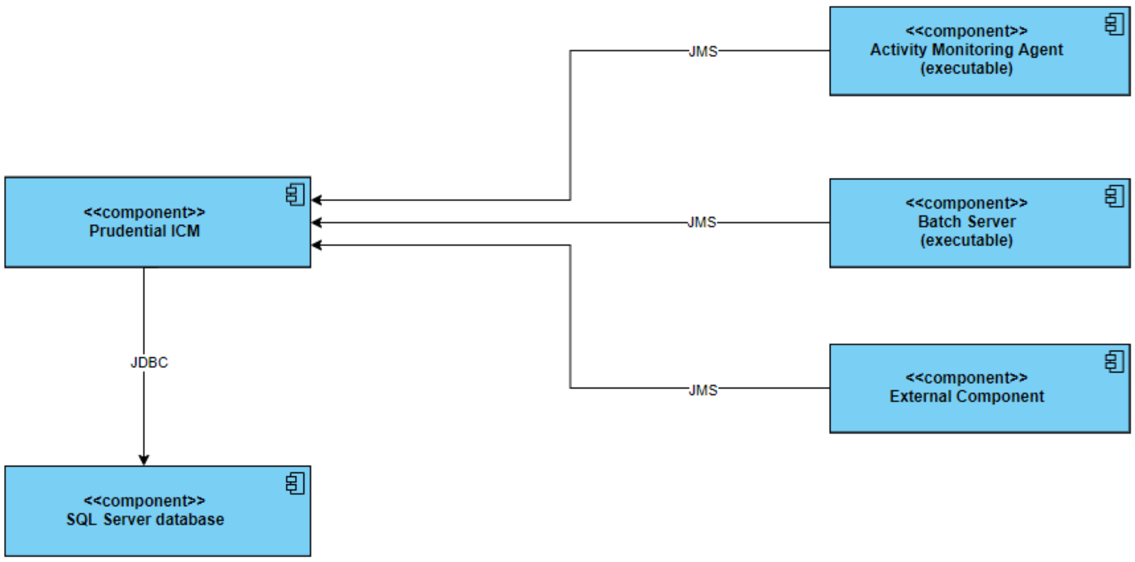
### 4.2.20 Activity Monitoring Framework (Error Handling)

Activity Monitoring Framework is designed to record batch runs activities during their execution to help users to track the progress of their processes and eventually analyze errors that happened during the runs.

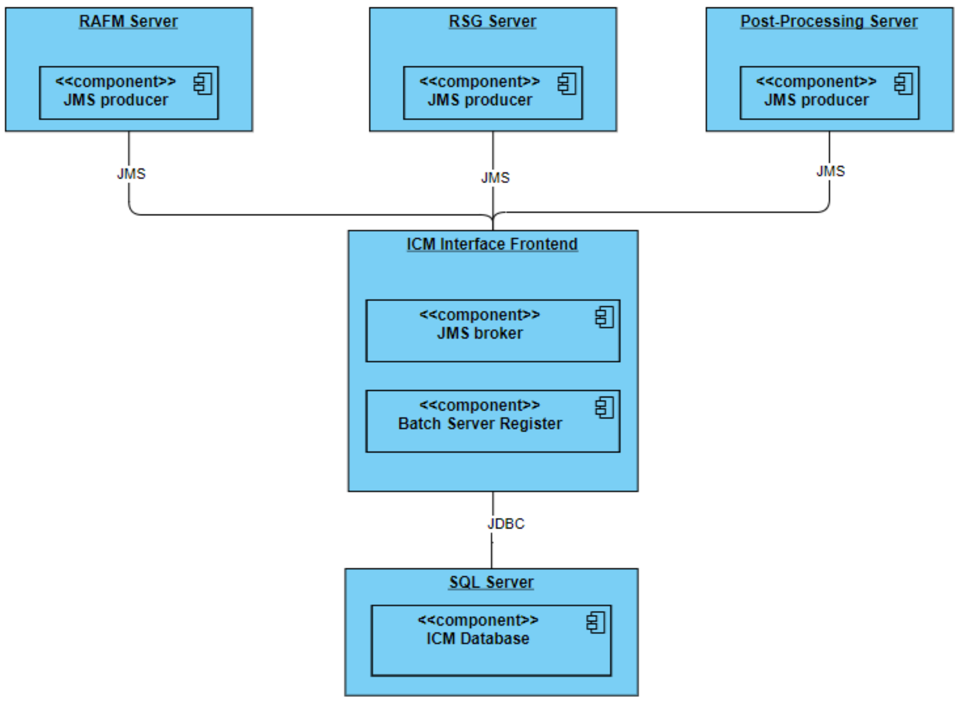
This framework consists of two main components - the server and the client components. The server component is a simple request processor that is responsible for receiving activity information and storing it into the activity repository, while client components are responsible for gathering information, processing it by transforming it into a standard format when needed, and sending it to the server.

The communication between the server and the clients is organized using JMS. The clients post *Activity Events* in a standard XML format (see section Activity Events for more details) and send them as the body of JMS message to the server that stores them into the *Activity Repository*.

The figure below shows the main components of this framework in the components diagram. The *ICM Prudential* component plays the role of the server, so it is responsible for receiving activity events from the clients and storing them in the SQL Server Database where the clients can be either *Activity Monitoring Agents* or *ICM Batch Server* or any other *External Component* that wants to send relevant activity data to be stored in the *Activity Repository*. It is important to note in this diagram that clients do not need to have direct access to the activity repository that resides in the SQL Server Database.



Regarding deployment, the server is deployed within *Prudential ICM* component in *Apache Tomcat*. *Agents* are installed on each box that needs monitoring. For machines that already have an *ICM Batch Server* installed, no agent needs to be installed, since the log file scanner agent is part of the batch server. On the other hand, for machines that have a service that should be monitored, but do not have an ICM Batch Server installed, a stand-alone agent needs to be installed. The figure below shows the deployment diagram for this framework.


RAFM Server
<<component>> JMS producer
RSG Server
<<component>> JMS producer
Post-Processing Server
<<component>> JMS producer
JMS
JMS
JMS
ICM Interface Frontend
<<component>> JMS broker
<<component>> Batch Server Register
JDBC
SQL Server
<<component>> ICM Database

### *4.2.20.1 Activity Events*

Messages posted to the server are called *Activity Events*. An Activity Event may describe the start or the end of a step in a batch run or an error or any other feedback that components want make available in the Activity Repository.

Activity Events are defined as follows:

| Name | Description | Required |
|---|---|---|
| Time | The date and time that the event happened. Format: ISO 8601 [YYYY]-[MM]-[DD]T[hh]:[mm] | Yes |
| host | The name of the server where the event was generated | Yes |
| source | The source of the event. The value of this field may be the log file name where the event was extracted from, or "ICM Batch Server" in the cases that the events are created by batch server, or any other source that may generate an event. | Yes |
| component | The name of the main component that generated the event | Yes |

| | | |
|---|---|---|
| subComponent | The name of the sub component that generated the event | Yes |
| environment | The name of the environment that generated the event | Yes |
| runId | The unique identifier of the Run | Yes |
| eventType | The type of the event. The possible values are [INFO, WARN, ERROR] | Yes |
| message | A text message that should be attached to the event | Yes |
| file | Log files related to the event | No |

As shown in the table above, Activity Events may also define additional files, for example files related to a run produced by a third-party component.

| **Name** | **Description** | **Required** |
|---|---|---|
| name | The name of the file. This name should identify the file on the server | Yes |
| value | The contents of the file | Yes |

See Appendix A for the schema definition for Activity Events.

#### *4.2.20.2 Activity Monitoring Agents*

Activity Monitoring Agents are specialized clients that scan components log files gathering relevant information and sending them to the server to be stored in the Activity Repository. These agents are deployed as simple Java processes that run in each machine that has log files that must be monitored.

The Activity Monitoring Agent is part of the Batch Server distribution, but is started as a light-weight process with a dedicated script. The script takes the configuration XML file and the file to be monitored as startup parameters. This way the configuration file, that defines the log format of a certain application, can be reused across application instances.

Both location and name of the log file being watched may define fields to be used to create Activity Events. A regular expression pattern can be specified in the configuration file in order to extract the relevant fields.

The running agent scans a particular file line by line. Matching patterns are defined in the configuration XML file. If a line matches a regular expression pattern, an Activity Event message is created and sent to the Frontend server over JMS. The Activity Monitor Agent uses the Batch Server’s configuration to locate the JMS broker.

This table summarizes the fields that can be extracted from a line or file name and that will be used to populate activity events.

| Name | Description |
|---|---|
| time | The date and time |
| component | The name of the component |
| subcomponent | The name of the sub component |
| environment | The name of the environment |
| runId | The run identifier |
| type | The event type. This should be one of the following values: INFO, WARN or ERROR |
| message | The message |

Any value defined in the table below may be used in this configuration.

The full schema is defined in Appendix A.

Here’s a sample configuration

```
<activity:log xmlns:activity="http://secondfloor.nl/ActivityEvent">


    <file-regex pattern=".*/(\w+)/(\w+)-.*-(\w+)\.log">

        <environment index="1" />

        <component index="2" />

        <runId index="3" />

        <subcomponent value="SBS" />

    </file-regex>


    <line-regex
pattern="(\S*\s\S*)\[.*\]\s+(DEBUG|INFO|WARN|ERROR)\s+(\S+)\s+(.*)$
">

        <time index="1" format="yyyy-MM-dd HH:mm:ss,SSS" />

        <type index="2">

            <replace match="DEBUG" value="INFO" />

        </type>

        <message index="4" />

    </line-regex>


</activity:log>
```

Let’s go through to sample file to highlight the configuration features.

The root element is **activity:log** and contains one optional child element **file-regex** and one or more **line-regex** elements.

These elements allow you to specify a regular expression in the match attribute. Based on matching groups, the matched contents are used to populate the Activity Events.

The **file-regex** element matches on the absolute file name of the log file being watched. (To maintain platform neutrality, forward slashes should be used as a path separator on Windows systems.) In the example, the first element has the **index** attribute set to 1 and thus will fill the ‘environment’ part in the Activity Event with the first matching group. The second group provides the ‘component’ part and ‘runId’ is taken from the third group. It is also possible to specify static values, as is shown in the **subcomponent** element. By setting the **value** attribute to “SBS”, this will effectively be set as the default value of the ‘subcomponent’ part. The matching rules defined in **file-regex** apply to all matched lines, unless overridden in a **line-regex** element.

The **line-regex** element functions in a similar way as the file-regex element. This element is used for each line scanned in the log file, using a “Unix tail” like mechanism. For each line that matches, an Activity Event is created. Note that there can be more than one line-regex elements, so you can define different triggers.

The time element specifies how the time should be parsed in the **format** attribute. The format is specified as being the same as the format used by Java’s SimpleDateFormat class.

Finally, any element can optionally contain **replace** tags. The **match** attribute is again a regular expression and matches against the captured group. In the sample file, DEBUG logging messages are sent as Activity Events with type INFO.

#### 4.2.21 RAFM Input Files Builder

One of the most important parts of ICM Interface is the integration with RAFM provided by WTW with RAFM Task Runner. RAFM Task Runner requires a number of input files which drive the execution and define the data to be used during an Assumption Set calculation. These input files follow different formats for each component. The RAFM Input Files Builder is the component responsible for the generation of these files in the different formats.

The LM Builder is triggered as part of the following use cases:

- ASM009 Run Assumption Set
- Export RAFM Project
- Run RSG Standalone
- Export RAFM Project for RSG Standalone

As described in Load Balancer, after the run is requested by the user and a batch server is assigned to execute that run, the batch server starts processing that run. At this point, during the execution of the run in the batch server is when the RAFM Input Files Builder is processed generating all the input files that are used in the calculation.

The RAFM Input Files Builder process is composed by two main steps: the first step retrieves and prepares data that is used in the second step where the actual files are written. For the first step, the RAFM Input Files Builder gathers all the information input by the user throughout the components of the Assumption Set, such as General Parameters files, Experience parameter files, Entity Structure, etc… preparing it to be used by the writers in the next step. During the first step, the files containing information used to write a result file are parsed. Also, this is the point where the data that depends on the full Assumption Set is validated. For example, the length of the name of the export attributes depends on the length of name of the node in the hierarchy where the Lite Model is assigned to.

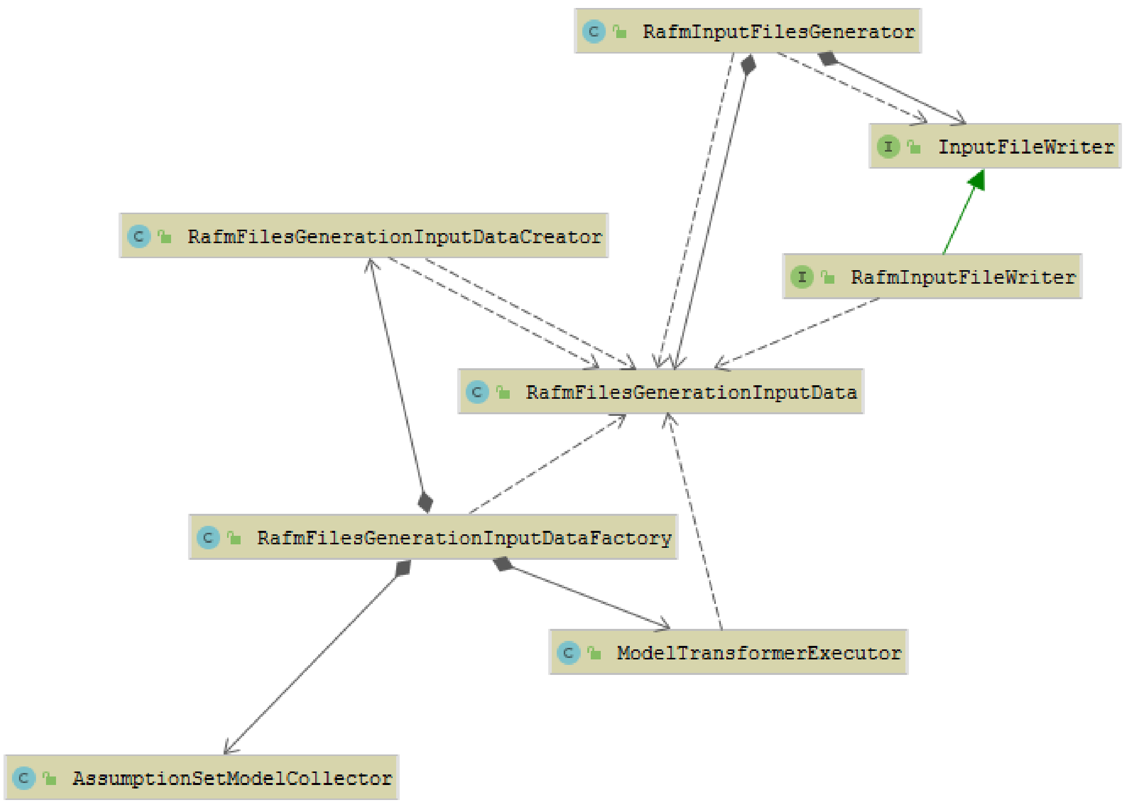
Having the data prepared, the RAFM Input Files Builder can start the second step, which is when the files are produced. This step consists of a collection of smaller processes called *writers*. Each writer is responsible for generating a type of file produced by the RAFM Input Files Builder.

There are different types of writers. Most of them are template-based writers that write a single file generated from a template, some of them generate multiple files (for example the writer responsible for writing RSG instruction set files from the scenario set or the writer responsible for writing the content of the RAFM project in the working folder). There are also conditional writers that are executed only if some condition is met, for example only for a given type of a Purpose of the scenario/assumption set.

Template-based writers generate files using *templates*. The templates define the structure of the file as well as the disposition of its content by defining where the data gathered in the first step is written in the file. With these templates and the data collected, the writer applies the data to the template and generates the resulting file. For example, there is a special writer that generates the `RAFM_ICM_Run_Settings.csv` file from the `RAFM_ICM_Run_Settings.csv.ftl` template.

There are about 30 writers in the RAFM Input Files Builder. All writers are split into several lists depending on the source of data and order of their execution, for example lite model data writers, aggregation rule data writers, general writers, synchronous writers, etc. Many writers that are not dependent on each other are executed simultaneously in multiple threads. They are called asynchronous writers. Some other writers, called synchronous writers, are executed only after all asynchronous writers have been executed. An example of synchronous writer is the static files writer.

The following class diagram shows the main classes that support RAFM Input Files Builder.


RafmInputFilesGenerator
InputFileWriter
RafmFilesGenerationInputDataCreator
RafmInputFileWriter
RafmFilesGenerationInputData
RafmFilesGenerationInputDataFactory
ModelTransformerExecutor
AssumptionSetModelCollector

### *4.2.21.1 Templates*

The templates are defined using Freemarker, which is *Template Engine* for Java. Freemarker has its own set of rules to define the templates, where the structure of the file can be mixed with some markers that are replaced with the right data when the data is applied to the template. For example:

```
This file is a ${fileType} file.
```

Would produce:

```
This file is a txt file.
```

If `fileType` is set to `txt` in the data applied to the template.

For more information about the structure of the templates, please refer to Freemarker documentation at http://freemarker.sourceforge.net/.

### *4.2.21.2 RAFM Input Files Builder Output files*

By applying data to the templates the writers generate the final input files for RAFM Task Runner. For each run, a new directory is created in this location and all files generated by the RAFM Input Files Builder are temporarily placed in this new directory.

The output directory is created using the following naming convention:

`<templates.working.dir>/<runId>` where runId is replaced by the run id.

For example, if `templates.working.dir` is

```
templates.working.dir=/tmp/rafm
```

and the run with ID 1012 is executed, the files generated by the RAFM Inputs Builder are stored in the directory:

```
/tmp/rafm/1012
```

Afterwards the batch server starts the Task Runner and waits for its completion. See RAFM Integration section for more details. At the end the RAFM Inputs Builder output directory (i.e. the working directory) is cleaned up.

#### *4.2.21.3 RAFM Inputs Builder Output Directory Structure*

For RAFM Runs the directory structure produced is as following:

- **Inputs**
  - **Lite Models** – LM Modelpoint and Lite Model input files
  - **RSG –** The RSG instruction set files (from the scenario set)
  - **Run Settings –** All other settings file (worker distribution files, ICM run settings file, RSG Control file, extracted nodes file and risk limit file)
- **Libraries -** External libraries that are used in the RAFM Project.
- **Obj/x64/Release -** Executables (RAFMCore.exe, Rafmdll.dll)
- **Obj/x64/Release/modelClass –** C++ files containing the source code for RAFM Project submodels
- **Scripts** – Scripts for post-processing of reports
- **RAFM_Parameters.rtk** – Task Runner parameters file
- **RAFM Project - ICM.msproj** – Project File
- **Code Manager file** (.mmf)

- Files referenced by the job:
    - Input Manager files (.ipm)
    - Output Manager files (.opm)
    - Run Manager files (.rmr)
    - Run Parameter Set files (.rps)
    - Output Parameter Set files (.ops)
    - Goal Seek Parameter Set files (.gsk)
    - Referenced Files file (.rfs)
    - Wildcards file (.wcd)

#### 4.2.22 Assumption Set Run Reports

An Assumption Set Run may have reports attached. The reports are generated by the RAFM and automatically linked to the Assumption Set Run, or manually uploaded by the user.

Assumption Set Reports are stored in the ICM Interface database.

For more details about the Assumption Set Reports, please refer to BP005.

#### 4.2.23 Stochastic assumption set runs with multiple shreds

ICM 7.0.0.0 introduces support for stochastic assumption set runs with multiple shreds. The user can trigger a stochastic run with base stochastic and/or multiple stochastic shreds. A single Task Runner execution (local or vGrid) will execute the run and produce reports for the base stochastic and/or the selected shreds.

Each assumption set run will create a main run ID (for the base stochastic or the first alphabetical shred if no base stochastic selected) and one individual run ID for each additional shred. Individual run IDs are linked to their correspondent main run ID.

Only the main run ID will be queued and sent to a batch server for execution.

After the main run is done, ICM will move shred reports from the main run output directory to the corresponding shred run output directory, replicate some of the other outputs to the shred runs output directories, and do post-processing of some reports in order to split them into multiple reports per shred.

The execution of an assumption run on the batch server consists of the following process steps:

Create report output directory for the main run ID

RAFM Task Runner stores the outputs of a given RAFM run in a separate directory, specified by setting the `OutputLocation` parameter in the RAFM Task Runner parameters file (RAFM_Parameters.rtk).

For a given RAFM run ICM sets this directory to the following value:

```
<rafm.outputs.dir> \ <geography> \ <mainRunID>
```

- The root directory is defined by a configuration property: `rafm.outputs.dir`
- The geography directory is the geography of the user that triggered the run
- runID is the run ID of assumption set run in ICM

Create report output directory for each shred run ID

For a given RAFM run with base and/or multiple shreds, ICM will create directories for each shred run ID as well:

```
<rafm.outputs.dir> \ <geography> \ <shredRunID1>

<rafm.outputs.dir> \ <geography> \ <shredRunID2>

..

<rafm.outputs.dir> \ <geography> \ <shredRunIDN>
```

Generate input files for the main run ID

The complete set of input files necessary for the RAFM run are prepared and stored in the working directory for the main run:

```
<templates.working.dir>/<mainRunID>
```

For more details, please refer to section 4.2.21

Execute the main run with Task Runner

The assumption set run is being sent to execution by calling RAFM Task Runner.

Replicate specified reports from the main run output directory to each shred run output directory

ICM will replicate some specific report files from the main run ID output directory to each shred run output directory. The list of files replicated across all shred run output directories is specified in section 8.1.

CSID_ST report post-processing

The previous step will make sure that the output directory for each run ID contains the CSID_ST.csv report file.

However, the CSID_ST.csv file produced by RAFM projects with WDF v1 contains the report data for all runs (base stochastic and/or multiple stochastic shreds).

To distinguish between the records from different run the file contains an extra column "Shred Type" that indicates the shred to which a given row belongs, or empty if the row belongs to the base stochastic run.

For assumption set runs using a WDF v1 project, ICM will go through each report output directory (for the main run ID and shred run IDs) and perform the following post-processing:

- Filter out records belonging to a different run ID
- Remove the "Shred type" column

This will ensure that each report output directory will contain a CSID_ST report with the correct format that contains the data for that run ID only.

Retrieve reports for the main run ID and store them in ICM

The report output directory for the main run will contain all reports produced for that run. ICM will retrieve a subset of those reports, store them in the ICM database and associate them with the corresponding run so that they are available for download from the ICM web interface.

<u>Retrieve reports for shred run IDs and store them in ICM</u>

The report output directory of the main run of a stochastic run with base stochastic and/or multiple stochastic shreds will contain reports belonging to different run IDs.

ICM will *move each report to its corresponding report output directory*, store it in the ICM database and associate it with the corresponding shred run ID.

<u>Stochastic extraction report post-processing</u>

The stochastic extraction report file produced by RAFM project with WDF v1 contains stochastic extraction reports for all runs (the base stochastic and/or multiple stochastic shreds).

To distinguish between the records from different run the file contains an extra column “Shred” (after the last report column) that indicates the shred to which a given row belongs, or empty if the row belongs to the base stochastic run.

For assumption set runs using a WDF v1 project, ICM will split the stochastic extraction report per shred (removing the extra column) and per node and allocate resulting reports to the corresponding run ID (the main run ID or one of the shred run IDs).

For assumption set runs using a WDF v0 project, ICM will split the stochastic extraction report per node and allocate resulting reports to the corresponding run ID.

<u>Retrieve log files and associate them with the main run ID and each shred run ID</u>

Log files that were replicated from the main run ID directory to all other run ID directories will be retrieved and associated with the corresponding run ID.

<u>Create manifest for the main run ID and for each shred run ID</u>

ICM will create a post-run manifest file for the main run ID and for each shred run ID. Their content will be virtually identical, the only difference being the run ID itself.

### 4.2.24 Run Cancellation

Cancellation is a process of stopping a Batch Run execution. Runs cannot be cancelled during their execution. Runs can be cancelled only before their execution i.e. when the status is Queued or Requested.

Assumption Set Run can be cancelled in the following cases:

- When a user selects an option 'Cancel Run'.

All runs can be cancelled from the Queue Balancer in the following cases:

- When a user selects an option 'Cancel Run'.

If a run is Queued or Requested then its cancellation process is as follows – the run status is immediately changed to 'Cancelled' so that this run will never go to any execution environment.

### 4.2.25 Security Model

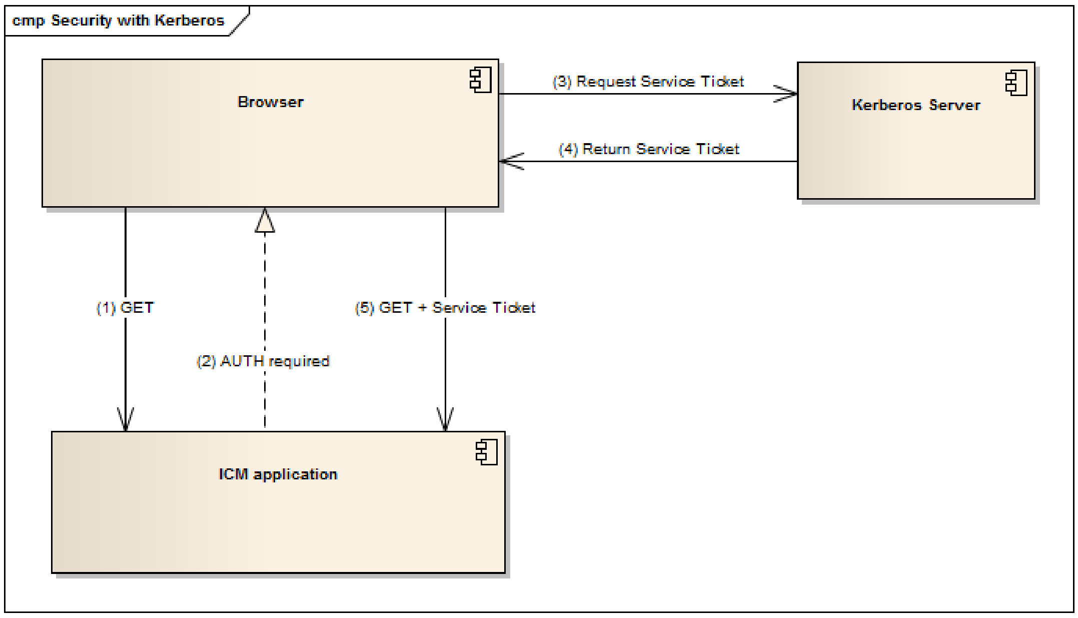
The model used for Authentication and Authorization is based on Principals, Roles and Permissions. There are two Authentication mechanisms supported: Kerberos based and property file based. The ICM application has a configuration that allows switching between those two types of authentication. Please refer to the System Configuration document for the details on how to apply this configuration.

On the running ICM application either Kerberos based or file based authentication is enabled. In the event of authentication failure no “fallback authentication” mechanism supported.

#### *4.2.25.1 Kerberos Authentication*

Kerberos is a standardized network authentication protocol, which is designed to provide strong authentication for client/server application.

Kerberos-based Authentication is implemented by adding the Spring Security framework with the Spring Security Kerberos extension that builds on top of Spring. With the Spring Security Kerberos Extension, users are authenticated against their web application just by opening the URL. There is no need to enter a username/password and no need to install additional software. That means that a user just enters the URL in the browser and he is automatically authenticated with his domain username. Here is a brief overview:



The Browser sends a GET request to the ICM application (1), which then returns that 'negotiate' authentication is required (2). The Browser will then ask the Kerberos Server to get a so called service ticket (3). The Browser then sends this service ticket, which proves the identity of the caller, and some additional things to the ICM application (5). After validating the ticket, based on some shared secret between the ICM application and the Kerberos server, it gets back the username.

For this to work, the ICM application needs to be registered at the Kerberos server and gets a service principal and a shared secret assigned. See System Configuration documentation for more information about how to generate a keytab file (shared secret) and how to make it available to the ICM application.

It is possible to have multiple Kerberos KDCs so that if one KDC failed then the browser failover to an alternative KDC to obtain a service ticket.

#### *4.2.25.2 Property file based authentication*

An alternative way of authentication is by using properties file. It can be useful for testing purposes or for environments that does not provide a Kerberos server. In this case users are configured in a property file. It can be also useful to provide users to be able to use the application before the real (Kerberised) users are defined in the application.

The following property files are used to define users:

- `users.properties` – for each user specifies user IDs, md5 password hash and user's roles

- `usergroups.properties` – contains a mapping of geographies (user group path) to users that belong to the geography

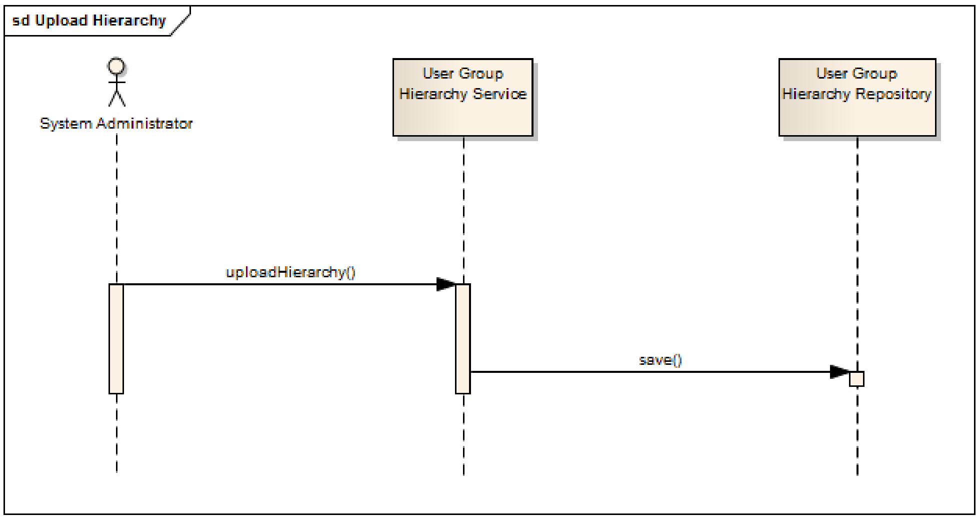
### *4.2.25.3 Authorization*

Authorization is done against ICM database where user groups and permissions are stored. In case of using properties file this authorization data takes precedence, so any of the authorization data defined in properties files is overwritten by the data stored in database.

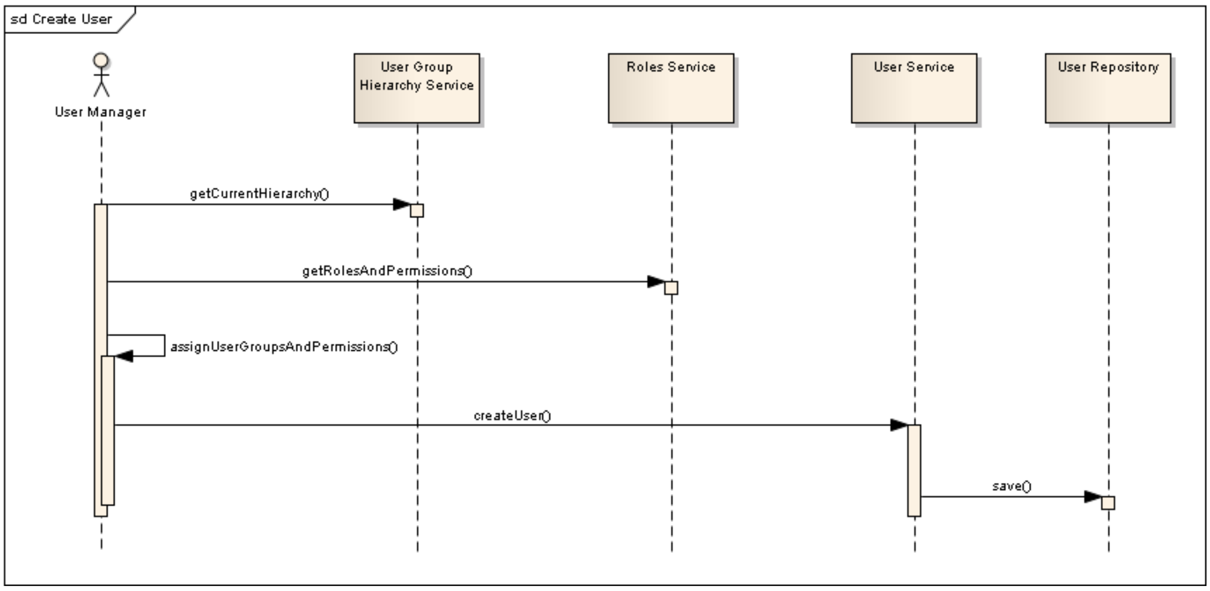
The list of all available roles is defined in the source code and used internally by the ICM Interface. Each role has a name, for example ROLE_ENTITY_STRUCTURES_LEVEL_1 and a description, for example ‘First level entity structures permissions’. The `roles.properties` file defines permissions for the roles and the roles hierarchy. The roles hierarchy defines the inheritance structure among roles. For example, ROLE_ENTITY_STRUCTURES_LEVEL_2 inherits all permissions of ROLE_ENTITY_STRUCTURES_LEVEL_1 and adds some other permission like the permissions to validate and reject entity structures.

```
ROLE_ENTITY_STRUCTURES_LEVEL_2=ROLE_ENTITY_STRUCTURES_LEVEL_1,

PERMISSION_SHARE_ENTITY_STRUCTURE,

PERMISSION_VALIDATE_ENTITY_STRUCTURE,

PERMISSION_REJECT_ENTITY_STRUCTURE,

PERMISSION_ASSIGN_GEOGRAPHIES_TO_ENTITY_STRUCTURE_NODES,

PERMISSION_ASSIGN_LEGAL_ENTITY_TO_ENTITY_STRUCTURE_NODES
```

The user group hierarchy is uploaded to the ICM interface by System Administrator.

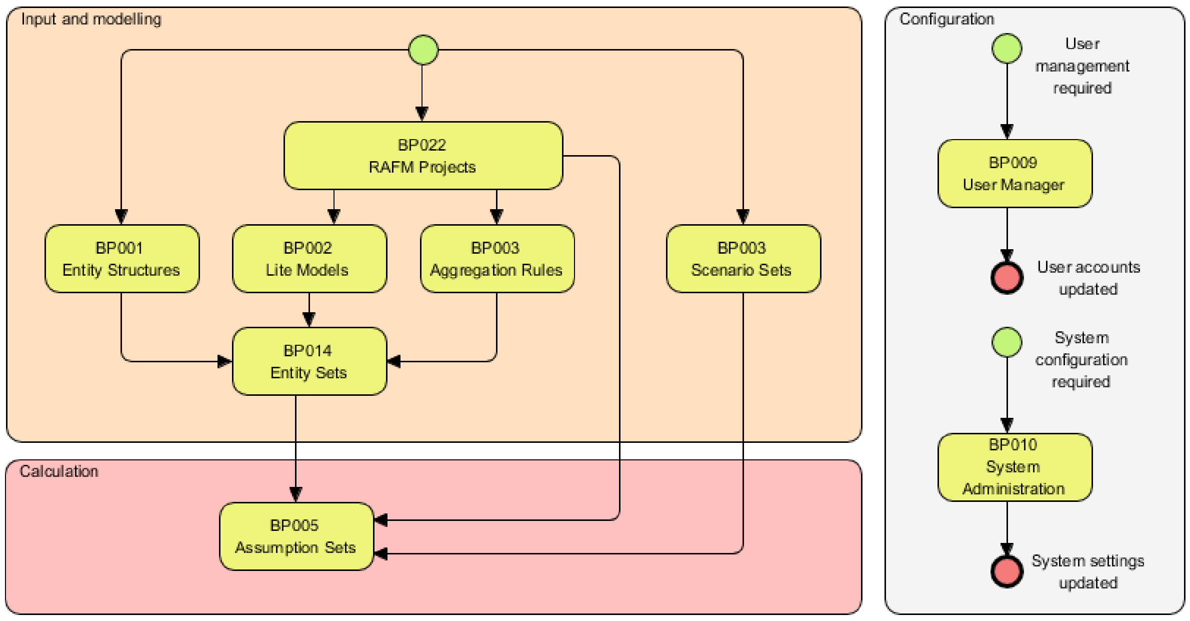


User Manager can create users in ICM interface and assign geographies, roles and permissions to the user. Every user can have multiple permissions in multiple user groups and/or multiple permissions in the same user group.



## 4.3 Business Logic

The figure below shows a high-level overview of dataflow in ICM Interface.



## 4.4 Interaction / Integration / Interfaces

### 4.4.1 RAFM Integration

Integration with RAFM is based on triggering a RAFM run through RAFM Task Runner.

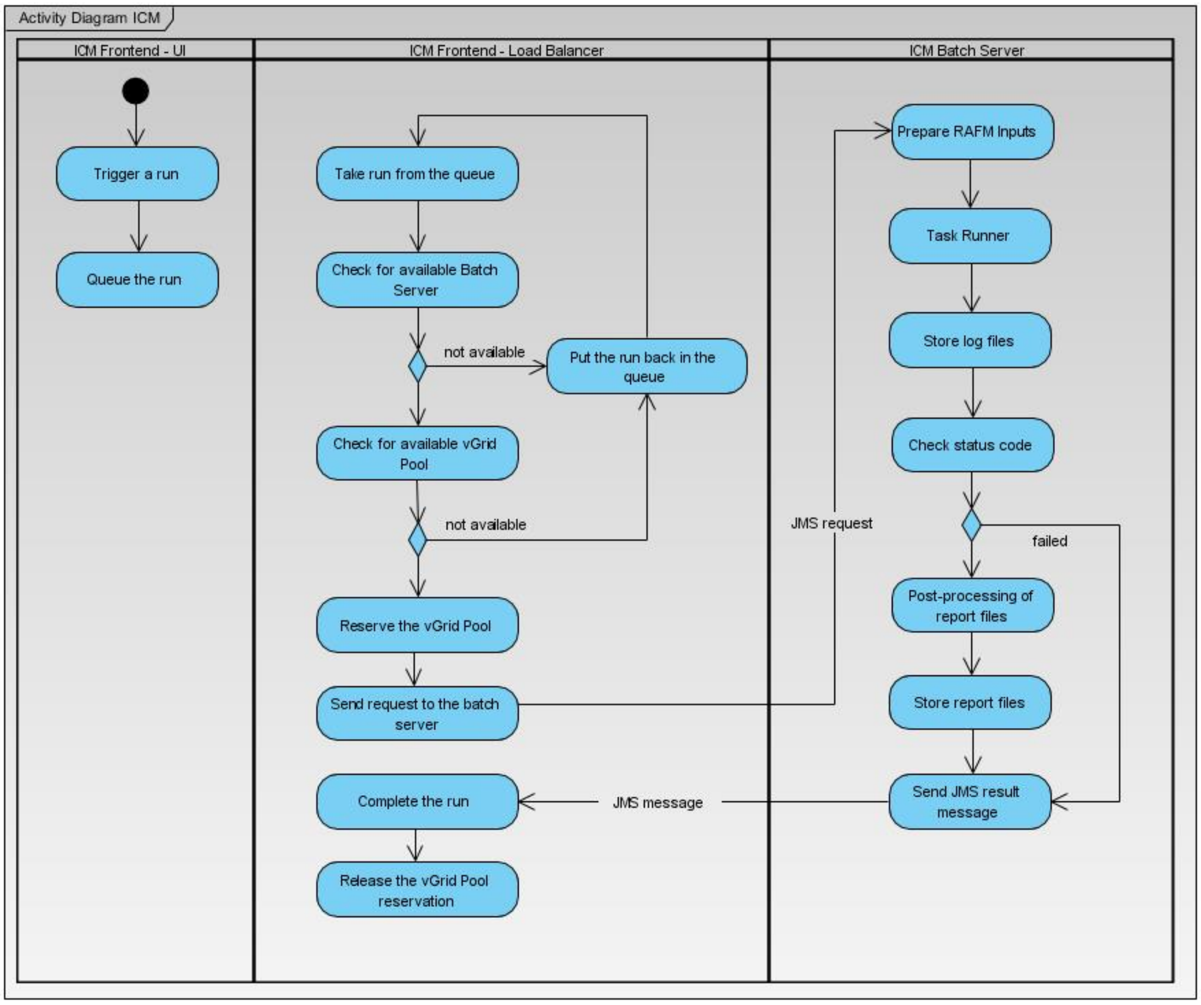
Before triggering RAFM jobs the RAFM Input Files Builder prepares the input data for RAFM Task Runner.

After the preparation of input files is done and all input files are available for RAFM Task Runner, the Batch Server creates a new process to run Task Runner and then waits for its execution.

After the execution is finished Task Runner will return with an exit code. The exit code indicates whether the run was successful or not. In case of an error the exit code gives more details about the source of the error. It is possible that the run failed because it was cancelled outside of ICM, directly in the vGrid tool. The Batch Server parses the log file produced by Task Runner in order to find out whether the run was cancelled outside of ICM or not.

At the end of the process, immediately after the run was processed with a status of Completed, Failed, Invalidated or Cancelled, a post-run manifest file is generated containing information regarding the assumption set selected, its assignments from the root node of the run to the lower nodes, scenario set assigned to it, run parameters, software versions of the IT systems used and run details. The post-run manifest file is stored in the ICM database, attached to the run and available to the user through download functionality.

The diagram below shows the whole process step by step:



### 4.4.1.1 *RAFM Task Runner*

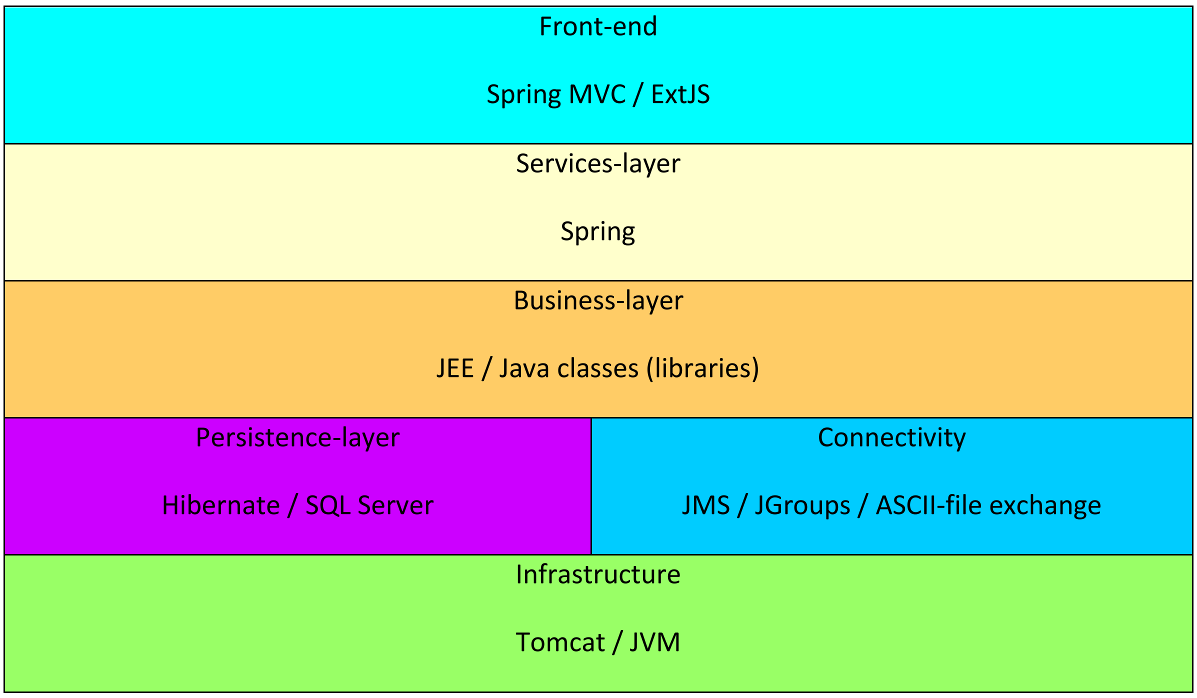
RAFM runs are executed through execution of the RAFM Task Runner tool. The Task Runner tool is documented in a separate document produced by WTW.

# 5 Physical System Implementation

The ICM Interface is designed as a multitier application consisting of one or multiple ICM Frontends and the ICM Batch Server Agents. The ICM Frontend is where the main components are deployed and is responsible for the orchestration of other components, while the Batch Server Agents are responsible for specific integration points.

## 5.1 ICM Frontend

The ICM Frontend application is divided into these layers:



Each of these layers represents a well-defined part of the application and communicates with one or more other layers. This kind of structure serves as a modularization which structures the implementation and the communication within the applications components.

Furthermore, the break-up makes maintenance and development more efficient.

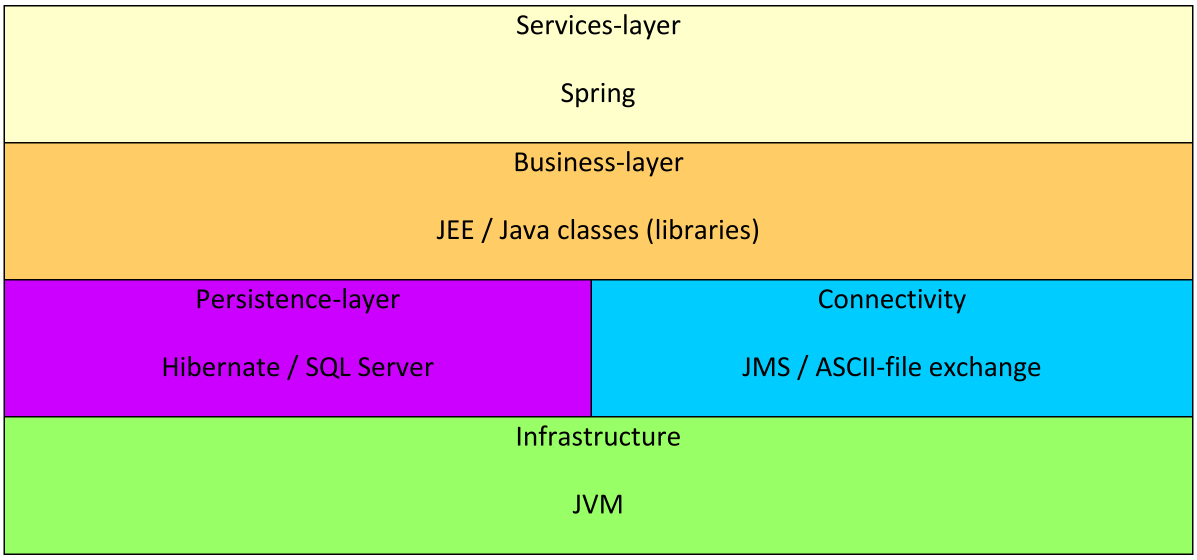
In the following table the role of each layer is explained in more detail:

| **Layer** | **Description** |
|---|---|
| Front-end | Using Spring MVC, frontend design follows the MVC pattern splitting implementation into models (data), views (display of data) and controllers (orchestration). Controllers and models use plain Java classes and views are designed with Java Server Pages technology using ExtJS JavaScript framework to render GUI components. Controllers respond to requests usually by |

| | |
|---|---|
| | delegating the execution to the service layer where business logic is implemented. |
| Service | The Service Layer exposes the main business logic of the application. It contains services to orchestrate execution flow and data repositories which are the interface for persistency operations. |
| Business | The business layer is where the domain model is defined. This layer encapsulates all the business specific logic. The libraries used by the application reside in this layer as well. |
| Persistence | The persistence layer is where the domain objects are persisted using ORM techniques. It uses Hibernate as the ORM implementation and SQL Server DB for its backend. |
| Connectivity | JMS and ASCII-file exchanges are used for integration with external systems.<br><br>Two options are possible for the JMS broker:<br><br>1) Embedded (only available when running a single instance of ICM Frontend)<br>2) External (should be used when running multiple instances of ICM Frontend)<br><br>The JMS broker (ActiveMQ) is used to provide connectivity and message storage/delivery functions. The message broker is also responsible for providing quality of service features, such as reliability, persistence, security and availability. |
| Infrastructure | Tomcat web server running on the Java Platform. |

## 5.2 ICM Batch Servers

The ICM Batch Servers are specialized integration services. They are responsible for translating the ICM requests into third part system specific calls. It reuses the business and persistence layers from the frontend and adds a new service layer that is responsible for the integration specific code. The overall architecture of the agents is as follows:



In the following table the role of each layer is explained in more detail:

| **Layer** | **Description** |
|---|---|
| Service | The Service Layer exposes the main business logic of the application. It contains services to orchestrate execution flow and data repositories which are the interface for persistency operations. |
| Business | The business layer is where the domain model is defined. This layer encapsulates all the business specific logic. The libraries used by the application reside in this layer as well. |
| Persistence | The persistence layer is where the domain objects are persisted using ORM techniques. It uses Hibernate as the ORM implementation and SQL Server DB for its backend. |
| Connectivity | JMS and ASCII-file exchanges are used for integration with external systems.<br><br>Two options are possible for the JMS broker:<br><br>1) Embedded (only available when running a single instance of ICM Frontend) |

| | |
|---|---|
| | 2) External (should be used when running multiple instances of ICM Frontend)<br><br>The JMS broker (ActiveMQ) is used to provide connectivity and message storage/delivery functions. The message broker is also responsible for providing quality of service features, such as reliability, persistence, security and availability. |
| Infrastructure | Java Platform |

## 5.3 Web Services

ICM Web Services consist of two endpoints: one is to check if a certain user has access to a certain RAFM run and the other one is to download the reports associated to a run.

In case of error, ICM will log the error details in the frontend log file where the request was sent.

### 5.3.1 check_geographical_permission

This endpoint is to check whether a user has access to a certain RAFM run.

URL: /check_geographical_permission/user/{userId}/run/{rafmRunIdString}

Input parameters:

- userId: the user id
- rafmRunIdString: the RAFM run ID

Response: ICM will return a custom response header “return-code” with two potential values:

- 0: the user has access to the RAFM run
- 1: the user does not have access to the RAFM run

### 5.3.2 get_reports

This endpoint is to download all the reports associated to a certain RAFM run.

URL: /get_reports/user/{userId}/runs/{rafmRunIdsString}

Input parameters:

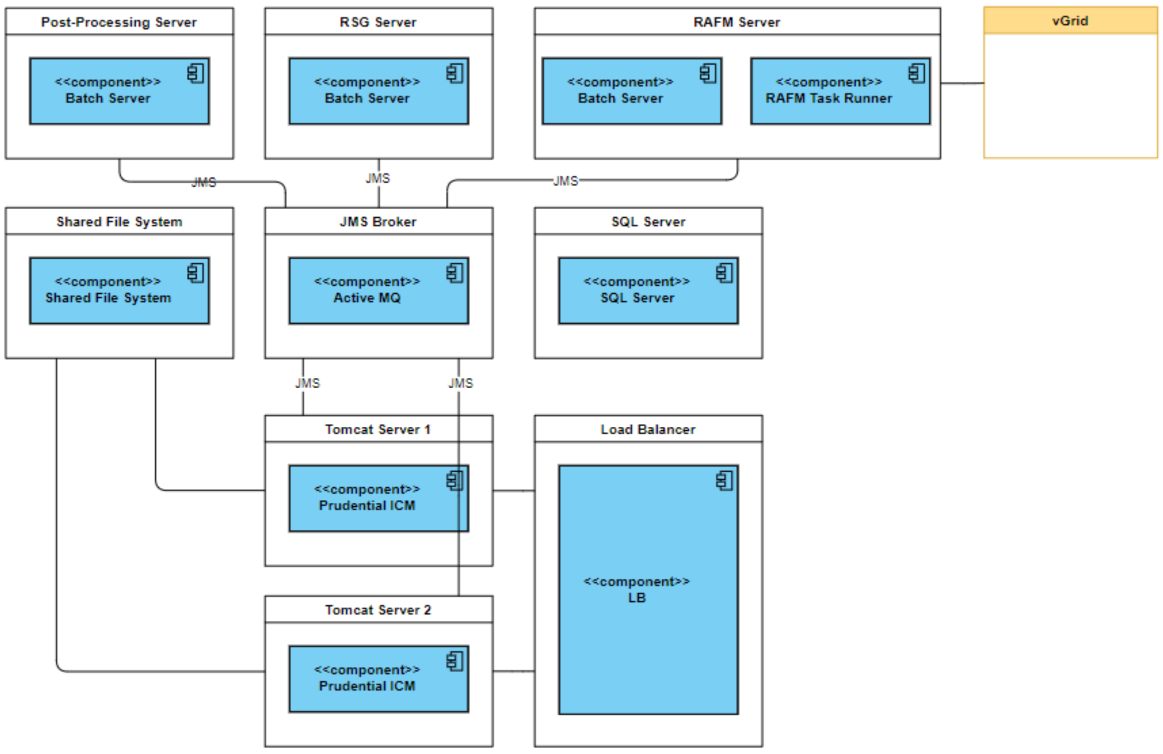
- userId: the user id
- rafmRunIdsString: string containing the RAFM run IDs separated with a dash. Example: 123-456-789

Response: ICM will return a zip file containing all the associated reports. In case of error, ICM will return one of the following error codes in a custom response header “return-code”:

- 201: the run was not found, ie the run ID was wrong
- 202: the user does not have geographical access to the run
- 203: the user was not found, ie the user id was wrong
- 205: unexpected error

## 5.4 Deployment

The following diagram shows how the system components are deployed, how they interact with each other and also gives an overview of the third party interfaces.



Some lines have been omitted from the diagram to avoid clutter. Please note that all batch servers communicate with the SQL Server database. In addition, the Post-Processing batch server and the ICM Tomcat server need to have access to a shared file system.

## 5.5 Configuration

For the information about how to install and configure the ICM Frontend and Batch Server applications please refer to the System Configuration documentation. There you can find:

- How to create the database schema, the application user and configure data sources.
- How to setup Kerberos security.
- How to configure logging utilities.
- And more

Please note that the ICM application doesn’t have a standardized set of error codes. In case of system failure or problems the log files will have to be studied. Most types of problems should result in a reasonably detailed failure notification in the relevant log file.

## 5.6 Clustering and Scalability

The ICM Frontend supports clustering.

Each ICM Frontend instance runs on its own Tomcat server. Session replication is implemented in the Tomcat server configurations i.e. the HTTP session is replicated across the ICM Frontends. The recommended and supported configuration is to set up a load balancer with "sticky session" configuration. This means that the user that is logging in to the system will always send its requests to the frontend assigned to his session and then keep using the same frontend server throughout the whole session. If this frontend dies, then the load balancer will assign another frontend to the user and send his requests there seamlessly and the user will not lose his session as it has been replicated

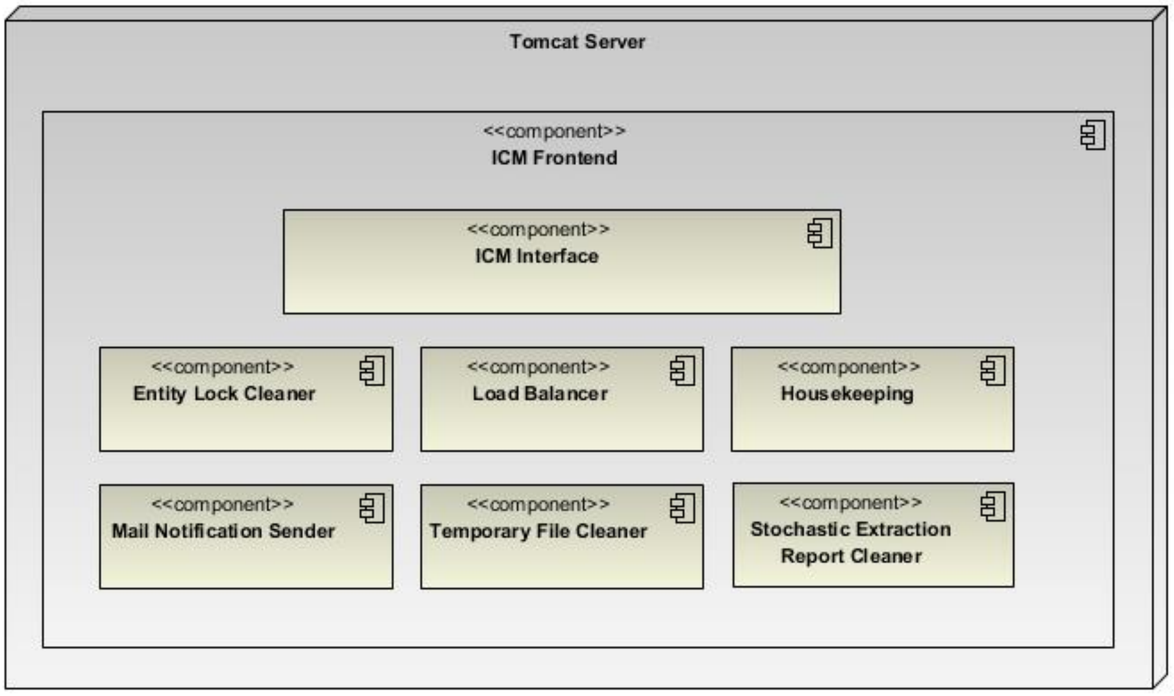
Multiple ICM Frontends can be part of a single cluster that will be aware of which ICM Frontend is available at a given moment.

The cluster will elect one ICM Frontend as leader. That leader will perform some services which are executed periodically in background threads as scheduled tasks. If the leader server dies, the cluster will elect another ICM Frontend as leader.

Here is the list of all scheduled tasks in the ICM Frontend application:

- *Entity lock cleaner* – a scheduled task for automatic unlock of the ICM components, it unlocks components whose lock reached some configurable timeout.
- *Load balancer* – a scheduled task that sends queued runs to available batch servers (for more details refer to Load Balancer)
- *Housekeeping* – a scheduled task to keep track of all registered batch servers (for more details refer to Housekeeping)
- *Mail notification sender* – scheduled task responsible for sending mail notification after a completion of a batch run.
- *Temporary file cleaner* – scheduled task that deletes error logs / consistency reports after a preconfigured period.
- *Stochastic extraction reports cleaner* – scheduled task that deletes the stored stochastic reports after a certain preconfigured period.

The following diagram shows the scheduled tasks in the context of the main process.



The Batch Server is a standalone application. It can be installed in as many servers as needed. There's no direct communication between the Batch Servers, they are completely independent of each other. But they all communicate with the leader Frontend which is responsible for distributing the load between those batch servers. Please refer to the section Load Balancer for more details.

## 5.7 Potential Performance Problems

From ICM 6.8 clustering of the Frontend application is fully supported which means that a Load Balancer in front of the ICM frontend server could provide horizontal scalability in case of big numbers of users concurrently accessing the application.

Assumption set tree building and assignments model, especially for big structures with nested entity sets, and any operations on them like search of a node with the given name or search of a node with the given lite model assigned.

Queuing mechanism could be more efficient. Currently the Load Balancer requests the Queue Manager to build the queue every time it runs (it runs periodically, for example, every 5 seconds).

There is no need to re-organize the queue every time assuming that it's not that dynamic. It would be more efficient to save the queue in the ICM database and modify it only when a new run request comes.

Bulk Upload of lite models and aggregation rules. In case of uploading very big amount of data (e.g. thousands of lite models) there might be OutOfMemory error because the result of parsing of the uploaded file is stored in the memory.

### 5.8 Single Points of Failure

The ICM Frontend running on Tomcat server supports clustering (see section Clustering and Scalability). Therefore, if we have multiple frontend servers and if one fails/stops the provided services and becomes unavailable, the users will still be able to access the ICM ICM Interface through another frontend server, provided there is a load balancer in front of the Tomcat servers that will take detect a failed node and redirect requests to another available ICM Frontend.

# 6 System & Service Operation

### 6.1 Release Processes

For the system and service operation we need to look into two different basic processes which define how releases are planned and delivered:

During an ongoing project – which is governed by an SoW – releases are scheduled according to the project plan which is aligned with the customer's needs for deployment and testing.

There might be preview releases containing part of the full scope in order to give Prudential the opportunity to test these parts early and to review the requirements. Preview releases need to be defined in an SoW or might be agreed during a project depending on circumstances. It needs to be agreed how this affects the start of UAT support.

The final delivery of the full scope will be followed directly by a UAT period supported by SecondFloor. This includes technical, test and business analyst capacity and is intended not only to fix issues but also to assist Prudential's staff with configuration and use of the system. The UAT period needs to be well structured: Both SecondFloor and Prudential need to agree on milestones during the UAT. One of the most important milestones is the end of testing by Prudential.

Work on a release which is in production might be covered by CR's or an SoW. In this case it will be handled as explained above. Otherwise, any bugs identified in a production releases will be fixed as

defined in the SLA agreement. All the details are defined in the SLA contract and are considered out of scope of this document.
However, usually for any work under SLA terms the client needs to define the scope and timeline for an SLA release. The timeline is subject to criticality and the agreed notice period for a release.

#### 6.1.1 Test Activities Performed by SecondFloor

Prior to any release SecondFloor performs the test activities as defined in the Master Agreement. This includes:

- Scripting and governance of all tests using Quality Centre which is provided internally by SecondFloor.
- Functional tests on the scope of a CR or a Statement of Work.
- Smoke testing.
- Regression test covering those parts of the application which might be affected by the performed changes.
- Integration testing using either mock-up or the actual third party application in case this one is available. Integration testing is also subject to the scope of the changes.
- End to end testing.

## 6.2 Release Types

SecondFloor delivers a full release package using Confluence. Prudential downloads the release and deploys it according to the installation notes which form part of the release notes. The release package consists of:

- All executables for all parts of the application
- System Configuration
- Release Notes
- Test Report

Patch releases are not usually delivered as this does most often not fit with Prudential’s internal processes. If requested by Prudential it needs to be agreed on a case to case basis. Therefore, it is obvious that such a patch release will contain only a minimal subset of a normal release and will depend on each case.

## 6.3 Maintenance

The maintenance of the ICM Interface itself is minimal but as the application orchestrates third party applications maintenance of these might influence the ICM Interface. The following cases should be considered:

- **Java Updates** – Updates within the same major release (Java 1.8) should not lead to problems. In case of updates to another major release SecondFloor needs to do an impact assessment and advise Prudential on necessary actions.
- **Tomcat Updates** – Any update on Tomcat should be impact assessed by SecondFloor as the used libraries might contain changes impacting the ICM.
- **OS Updates** – The update of the operating system on machines on which layers of the ICM are running might impact the ICM. As long as the Java installation, Tomcat and the infrastructure components needed for the ICM to communicate does not change – no change should be necessary to the ICM. This should be assessed by responsible administrator. Whenever there might be changes to the required components and services for the ICM Prudential needs to request and impact assessment from SecondFloor.
- **Updates to third party applications** – Changes to any interface used by the ICM Interface need to be communicated to and impact assessed by SecondFloor. In most cases a change request needs to filed and scheduled.
- **Database** – The administration of the database including backups, monitoring of performance and the standard administrative tasks, e.g. rebuilding statistics, dealing with file fragmentation and log file management. These tasks are the responsibility of Prudential's DBAs.

It is the responsibility of Prudential to assess all changes to machines, software and infrastructure used by the ICM and inform SecondFloor about all changes which might impact the ICM and any of the interfaces presented or used by the ICM. In addition to that it is the responsibility of Prudential to test such changes appropriately.

# 7 Appendix A – XML Schema Definitions

## 7.1 Activity Monitoring

This is the XSD that defines the Activity Monitoring message format.

```
<?xml version="1.0" encoding="UTF-8"?>

<schema

        xmlns="http://www.w3.org/2001/XMLSchema"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:activity="http://secondfloor.nl/ActivityEvent"

        xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"

        targetNamespace="http://secondfloor.nl/ActivityEvent"

        jaxb:version="2.1"

        xsi:schemaLocation="http://java.sun.com/xml/ns/jaxb
http://java.sun.com/xml/ns/jaxb/bindingschema_2_0.xsd ">


    <element name="activityEvent"
type="activity:ActivityEventMessage"/>


    <complexType name="ActivityEventMessage">

        <sequence>

            <element name="time" type="dateTime"/>

            <element name="host" type="string"/>

            <element name="source" type="string"/>

            <element name="component" type="string"/>

            <element name="subcomponent" type="string"/>
            <element name="environment" type="string"/>

            <element name="eventType"
type="activity:ActivityEventType"/>

            <element name="jobId" type="string"/>

            <element name="message" type="string"/>

            <element name="file" type="activity:LogFileContents"
minOccurs="0" maxOccurs="unbounded"/>

        </sequence>

    </complexType>


    <simpleType name="ActivityEventType">

        <restriction base="string">

            <enumeration value="INFO"/>

            <enumeration value="WARN"/>

            <enumeration value="ERROR"/>

        </restriction>

    </simpleType>


    <complexType name="LogFileContents">

        <simpleContent>

            <extension base="string">

                <attribute name="name" type="string"/>

            </extension>

        </simpleContent>

    </complexType>
```

```
</schema>
```

## 7.2 Activity Monitor Agent Configuration

This XSD defines the configuration file format for the Activity Monitor Agent

```
<?xml version="1.0" encoding="UTF-8"?>

<schema

        xmlns="http://www.w3.org/2001/XMLSchema"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:activity="http://secondfloor.nl/ActivityEvent"

        xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"

        targetNamespace="http://secondfloor.nl/ActivityEvent"

        jaxb:version="2.1"

        xsi:schemaLocation="http://java.sun.com/xml/ns/jaxb
http://java.sun.com/xml/ns/jaxb/bindingschema_2_0.xsd ">


    <element name="log" type="activity:LogConfiguration"/>


    <complexType name="LogConfiguration">

        <sequence>

            <element name="file-regex" type="activity:RegexMatcher"
minOccurs="0"/>

            <choice minOccurs="0" maxOccurs="unbounded">

                <annotation>

                    <appinfo>

                        <jaxb:property name="lineMatchers"/>
                    </appinfo>

                </annotation>

                <element name="line-regex"
type="activity:RegexMatcher"/>

                <element name="line-csv"
type="activity:CsvMatcher"/>

            </choice>

        </sequence>

    </complexType>


    <complexType name="ActivityMatcher">

        <all>

            <element name="time" type="activity:DateMatcher"
minOccurs="0"/>

            <element name="component" type="activity:TextMatcher"
minOccurs="0"/>

            <element name="subcomponent" type="activity:TextMatcher"
minOccurs="0"/>

            <element name="environment" type="activity:TextMatcher"
minOccurs="0"/>

            <element name="jobId" type="activity:TextMatcher"
minOccurs="0"/>

            <element name="type" type="activity:TextMatcher"
minOccurs="0"/>

            <element name="message" type="activity:TextMatcher"
minOccurs="0"/>

        </all>

    </complexType>
    <complexType name="RegexMatcher">
        <complexContent>
            <extension base="activity:ActivityMatcher">
                <attribute name="pattern" type="string"/>
            </extension>
        </complexContent>
    </complexType>

    <complexType name="CsvMatcher">
        <complexContent>
            <extension base="activity:ActivityMatcher">
                <attribute name="separator" type="string"/>
                <attribute name="quoted" type="boolean"/>
            </extension>
        </complexContent>
    </complexType>

    <complexType name="TextMatcher">
        <sequence>
            <element name="replace" type="activity:Replacement"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="index" type="int"/>
        <attribute name="value" type="string"/>
    </complexType>
    <complexType name="Replacement">
        <attribute name="match" type="string"/>
        <attribute name="value" type="string"/>
    </complexType>

    <complexType name="DateMatcher">
        <attribute name="index" type="int" use="required"/>
        <attribute name="format" type="string" use="required"/>
    </complexType>

</schema>
```

# 8 Appendix B

## 8.1 List of reports replicated to shred run output directories

| File name | Post processing |
|---|---|
| csid_st.csv | Yes |
| risk_measure_output_001.csv | |
| risk_measure_output_002.csv | |
| risk_measure_output_003.csv | |
| risk_measure_output_004.csv | |
| risk_measure_output_005.csv | |
| risk_measure_output_006.csv | |

| | |
|---|---|
| risk_measure_output_007.csv | |
| risk_measure_output_008.csv | |
| risk_measure_output_009.csv | |
| risk_measure_output_010.csv | |
| risk_measure_output_011.csv | |
| risk_measure_output_012.csv | |
| risk_measure_output_013.csv | |
| risk_measure_output_014.csv | |
| risk_measure_output_015.csv | |
| risk_measure_output_016.csv | |
| risk_measure_output_017.csv | |
| risk_measure_output_018.csv | |
| risk_measure_output_019.csv | |
| risk_measure_output_020.csv | |
| risk_measure_output_021.csv | |
| risk_measure_output_022.csv | |
| risk_measure_output_023.csv | |
| risk_measure_output_024.csv | |
| risk_measure_output_025.csv | |
| risk_measure_output_026.csv | |
| risk_measure_output_027.csv | |

| | |
|---|---|
| risk_measure_output_028.csv | |
| risk_measure_output_029.csv | |
| risk_measure_output_030.csv | |
| risk_measure_output_031.csv | |
| risk_measure_output_032.csv | |
| risk_measure_report.csv | |
| read_in_parameters.csv | |
| rafm_lite_model_output.csv | |
| rsg_results_report.csv | |
| rsg_validation_sample.csv | |
| stochastic extraction.csv | Yes |
| stochastic extraction_unsorted.csv | |
| shred_binary_report.csv | |
| shred_list_output.csv | |
| sort_order.csv | |
| AuditReport.pdf | |
| Job Info.txt | |
| Output Info.txt | |
| Runlog.txt | |
| Top Model_Generate Scenarios.txt | |
| Top Model_Reevaluation of Scenarios.txt | |

| | |
|---|---|
| Ranking_Perform Ranking.txt | |
| Task Runner Runlog.txt | |
| vGrid Job Info.txt | |
| WorkerSummaryReport.csv | |