BalticLSC Software User Requirements Specification

Detailed list of requirements for the BalticLSC Software

Version 1.0

Priority 1: Innovation

Warsaw University of Technology, Poland
RISE Research Institutes of Sweden AB, Sweden
Institute of Mathematics and Computer Science, University of Latvia, Latvia
EurA AG, Germany
Municipality of Vejle, Denmark
Lithuanian Innovation Center, Lithuania
Machine Technology Center Turku Ltd., Finland
Tartu Science Park Foundation, Estonia

BalticLSC Software User Requirements Specification

Detailed list of requirements for the BalticLSC Software

Work package	WP3
Task id	A3.3
Document number	O3.3
Document type	Requirement Specification
Title	BalticLSC Software User Requirements Specification
Subtitle	Detailed list of requirements for the BalticLSC Software
Author(s)	Agris Šostaks (IMCS), Sergejs Rikačovs (IMCS), Bartosz Sawicki (WUT), Michał Śmiałek (WUT), Kamil Rybiński (WUT)
Reviewer(s)	Daniel Olsson (RISE), Maj Held (Vejle)
Accepting	Michał Śmiałek (WUT)
Version	1.0
Status	**Final Version**

EUROPEAN REGIONAL DEVELOPMENT FUND
EUROPEAN UNION

History of changes

Date	Ver.	Author(s)	Change description
11.07.2019	0.1	Agris Šostaks (IMCS)	Initial structure.
24.09.2019	0.2	Agris Šostaks (IMCS)	Initial content.
02.10.2019	0.3	Agris Šostaks (IMCS)	Domains and user study added.
02.10.2019	0.4	Agris Šostaks (IMCS)	Updated models. Added headers for Sections 3, 4.
04.10.2019	0.5	Agris Šostaks (IMCS)	Non-functional requirements added.
08.10.2019	0.6	Agris Šostaks, Sergejs Rikačovs (IMCS)	Executive summary, introduction, conclusion, and user stories added.
10.10.2019	0.7	Agris Šostaks (IMCS), Bartosz Sawicki (WUT)	The second user-study added.
13.10.2019	0.8	Agris Šostaks (IMCS)	Formatting.
28.10.2019	1.0	Agris Šostaks (IMCS)	Added the third user study. Included reviewers' suggestions.

Executive summary

The overall aim for the Baltic LSC activities is developing and providing a platform for truly affordable and easy to access LSC Environment for end-users to significantly increase capacities to create new innovative data-intense and computation-intense products and services by a vast array of smaller actors in the region.

Current BalticLSC Software User Requirements Specification document is based on end-user requirements obtained during Baltic LSC project's international and local workshops, on individual meetings with potential cooperation partners and end-users, as well as on technical workshops with participation of external experts and led mainly by the project's technology partners from - Warsaw University of Technology (WUT), Institute of Mathematics and Computer Science, University of Latvia (IMCS), RISE Research Institutes of Sweden AB (RISE) . All workshops and meetings are held during Q1-Q2-Q3 2019.

BalticLSC Software User Requirements Specification document contains technical user requirements specification for BalticLSC Software. It is intended to use by BalticLSC technical partners responsible for the design and development of BalticLSC Software.

Table of contents

1. Introduction

1.1 Objectives and scope

The objective of the document is to describe the requirements for BalticLSC Software. The goal is to have the BalticLSC Software requirements specification which is usable for the BalticLSC Software Design. Since the BalticLSC Software development is being developed using agile methodology (Scrum) and the detailed requirements are developed during development sprints, this document is considered a “live” document and is a subject of changes after every sprint (approximately once per month).

The document describes the potential users, domains and user-studies for Baltic LSC Software. The document contains also Baltic LSC Software conceptual model (as UML class diagrams), functional requirements (as UML use-case model and usage scenarios), and non-functional requirements in the level of details required for the first sprints. Thus, the requirements specification covers full-functionality in the low details, and the usage scenarios for the first sprints have been described in details.

1.2 Relations to Other Documents

The current BalticLSC Software User Requirements Specification document is based on the information contained in the vision documents: BalticLSC Environment Vision (Baltic LSC Output 3.1), BalticLSC Platform Architectural Vision (Baltic LSC Output 4.1), and BalticLSC Software Architectural Vision (Baltic LSC Output 5.1).

1.3 Intended Audience and Usage Guidelines

This Requirements Specification is the Output 3.3 as described in the Baltic LSC Application Form and intended for internal use within BalticLSC consortium as basis for future software development activities as well as for reporting purposes for local First Level Control and Baltic Sea Region Program.

1.4 Notation Convention

Simplified versions of UML diagrams (Use-case, Class). Free-form diagrams.

2. Target Areas

In this chapter we describe the potential user groups and application domains of the BalticLSC Software (Environment). Next we describe several user-studies in more detail. The goal is to fix several user-studies – a simple user study which requires minimal functionality from both BalticLSC Software and Platform and will be implemented during first periods of development activities and more complex user studies that require more advanced features and will be implemented at the end of project.

2.1 Users

As it has already been identified in O3.1 BalticLSC Environment Vision and O5.1 BalticLSC Software Architectural Vision the main BalticLSC Software users can be divided into four categories: end-user, supplier (resource provider), developer and administrator:

Name	**End-user**
Description	End-users are using BalticLSC Environment to perform large scale computing with provided ready-to-use applications they can find in the BalticLSC Appstore. They are paying for used computing resources and applications with credits purchased at BalticLSC Environment.
Name	**Supplier (Resource provider)**
Description	Suppliers provide computing resources to the BalticLSC Network. In exchange, they receive credits from end-users for computations performed on provided re-sources.
Name	**Developer**
Description	Developers provide applications and modules to the BalticLSC AppStore. In ex-change for use of their applications and modules, they receive credits from the end-users.
Name	**Administrator**
Description	Administrators manage computing resources within the BalticLSC Network. Administrators are also responsible for approving new resource providers, new modules and applications for the BalticLSC Environment.

2.2 Application Areas and Domains

The goal of the BalticLSC project is to bridge the gap between problem owners (End-users), problem solvers (Developers) and resource owners (Suppliers). Thus, the BalticLSC Network solves three main problems:

- Lack of appropriate computation power for end-users – it allows finding appropriate re-source.
- Lack of appropriate problem-solving expertise/tool for end-users – it allows finding an ex-pert/application that solves the end-users problems.
- Insufficient computation resource utilization for suppliers – it allows offering computation resources when the resource is not used by the owner.

BalticLSC Network should allow providing resources of wide spectrum – starting from a laptop and ending with High-Performance Computing (HPC) machines. The requirements for hardware are de-scribed in Outputs O4.1, O4.2, and O4.3 which define vision, components, and requirements for Bal-ticLSC Platform.

BalticLSC Software in general and Computation Application Language, in particular, are targeting a wide range of application domains. Thus, any problem domain which requires computation power which exceeds the capabilities of a single laptop is within consideration. User studies from different domains have been summarized in O3.1 as a result of requirements workshops with the participation of potential stakeholders of the BalticLSC Network. We have identified several areas of application of the BalticLSC Environment. These areas (types of problems) are cross-cutting several application domains.

Area	**World (e.g. Physics, Process) Simulation/Modelling/Rendering**
Description	Real-world is incredibly complex. Thus simulation of natural phenomena, like, weather or liquids, involves lots of computations. Typically these types of problems have also lots of data and parameters involved. Typically simulation problems are **highly parallelizable** and **requires lots of computation power and data**. They are solved on HPC machines or distributed over large network of less powerful machines. Weather forecasting, molecular modelling – a technique to mimic molecular structure and interactions between 3D molecules, which may be used, e.g., in medicine design or quantum chemistry, are just a few examples which makes predictions, models the real world. Another in some sense similar problem area is the 3D rendering of movie/game/virtual reality scenes. It is a kind of simulation, but not of the real world, but of imaginary – digital. It also requires lots of computation power and is highly parallelizable.
Domains	Weather Forecasting, Chemistry, Geology, Entertainment (e.g., Gaming, Movie making), Virtual Reality
Area	**Optimization**
Description	Finding optimal or at least feasible solutions in many domains is a hard problem. Typically it requires **lots of computation power**, but it may not involve lots of data, e.g., in the transportation area the route planning (optimization) is such a problem. It **requires very specific data** – road network, traffic statistics, information on vehicles and destination. The computation of the route involves gradual improvement of the solution which takes time and resources. We have identified also problems and potential in other domains from transportation, e.g., finding the optimal dose for radiation in cancer treatment, optimizations in construction and/or engineering, optimization of processes for downtime minimization, optimization for risk reduction in different environments.
Domains	Transportation, Medicine, Process Management, Engineering
Area	**Image/Video/Speech/Text analysis**
Description	The recent advances in the area are remarkable. The reason behind it is the possibility to build complex systems automatically using machine learning algorithms. **Training** such systems requires both, **lots of data and lots of computation power**. Luckily, lots of images/videos/audios/texts are available and it is possible to build systems which analyse automatically, e.g., satellite data in order to make maps, to evaluate the state of the forests or agriculture lands, or patient examination images in order to help to set a diagnosis. Media is a domain where lots of work is done. Video is automatically transcribed, translated and even voiceover. There are systems that help to search within video or audio. C**omputation power** needed by such systems **is dependent on the amount of data** (images, video, audio, text) to be analysed.
Domains	Space, Agriculture, Medicine, Media
Area	**Data analysis (in general)**

Description	Since the amount of data stored in the data stores is huge and it grows very fast, the demand for computation power for the analysis of the data is also growing. Data analysis is a wide area, but it also has well-defined traditions, methods, and algorithms that are available in the form of software code libraries. It allows us to build module libraries within the BalticLSC Environment which are well re-usable over a wide range of problem solutions. Some of the domains where data analysis problems have been identified are banking, e.g., fraud detection is becoming an issue, engineering, e.g., vibration data processing in order to detect potential damage in mechanical systems.
Domains	Banking, Engineering

Of course, this list of areas and domains is not exhaustive and can be (and will be!) supplemented with new items. However, in these areas we have identified potential user-studies and SME-s or organizations which may be involved in the project.

It should be noted that many problem areas/domains involve **machine learning (Deep Learning) steps** that are computationally and data intensive. These steps by itself can be considered as a large area of application of the BalticLSC Network.

In many user-studies, the computationally intensive step is just one of many in the complete process. Thus, the end-user wants to delegate to the BalticLSC Network just one step, preferably integrated into their existing software. That means that BalticLSC Software should provide also **capabilities for automatic usage of the system by external software**.

Taking into account the observed areas and domains of application, we consider that Computation Application Language (CAL) should be universal. It means **CAL should provide universal data and control structures**. Domain-specific language structures can be implemented as particular modules or module libraries.

2.3 User-Studies

This section describes two user-studies – a simple user study that requires minimal functionality from both BalticLSC Software and Platform and will be implemented during the first periods of development activities and more complex user study which requires more advanced features and will be implemented at the end of the project.

2.3.1 Video processing: person's appearance detection

Video Processing: Person's Appearance is a task when one should recognize a known person(s) in a video and find time-frames when the person(s) has appeared. The user study comes from Latvia, Riga – a start-up (SME), SIA "Noscos", offers the face recognition service for internal media libraries. It allows creation of the image database and training the system to recognize persons who appear in the database. When the persons database is ready, the system allows to process videos and tag the timeframes where a person appears.

The system is used to manage the image database (archive) and also to manage the processed videos. The system is used also to search in the image archive and videos even if the previously unrecognized person appears! The system allows also to create connection graphs based on the persons' appearance in the videos. Thus, there are lots of functionality the existing system provides, however, the computation-intensive step is only the video processing step. This step is the only one Noscos wants to delegate to an external service. The reason behind it is the lack of their own computation resources.

The video processing step has a video and person database as input and produces the list of timeframes the persons from the database appear in the video (see. Fig. 1).

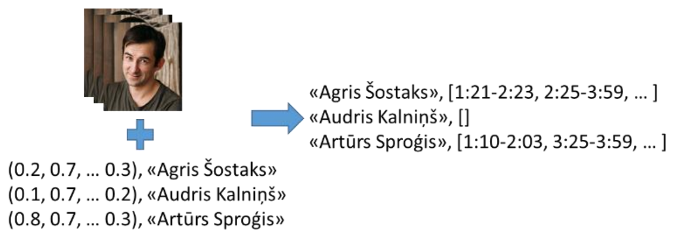

«Agris Šostaks», [1:21-2:23, 2:25-3:59, …]
«Audris Kalniņš», []
«Artūrs Sproģis», [1:10-2:03, 3:25-3:59, …]
(0.2, 0.7, … 0.3), «Agris Šostaks»
(0.1, 0.7, … 0.2), «Audris Kalniņš»
(0.8, 0.7, … 0.3), «Artūrs Sproģis»

Figure 1 Parameters and the result of the Video processing

The Noscos system (see Fig. 2) uses a separate internal micro-service (Docker container) which implements the desired functionality.

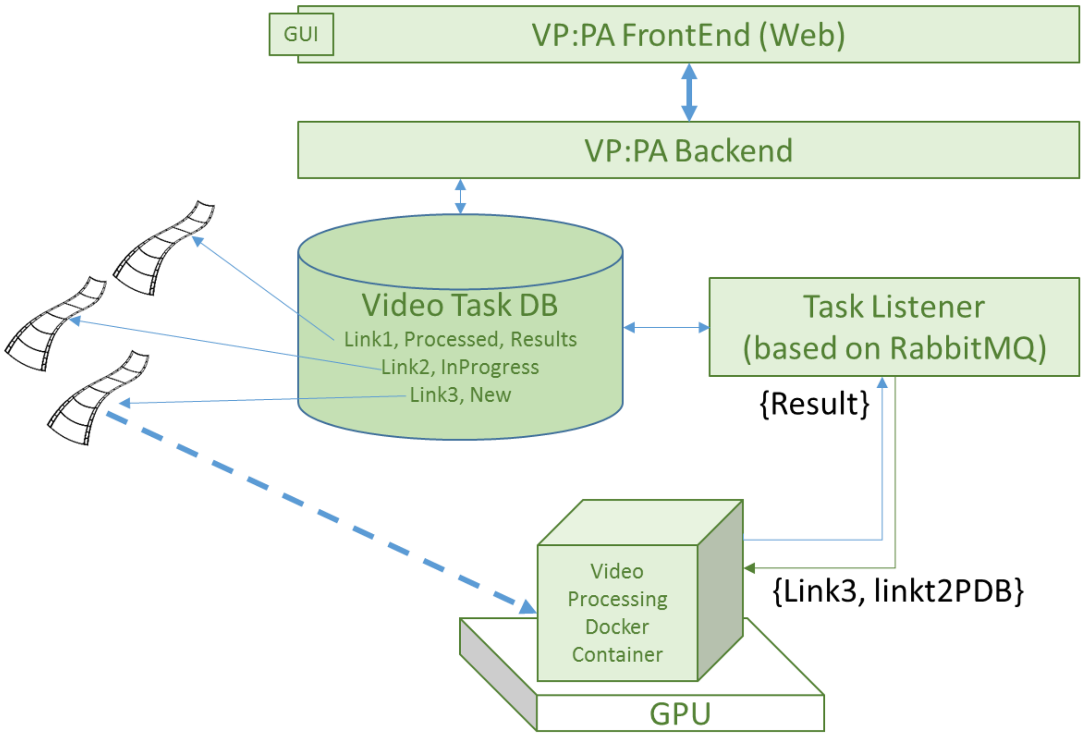


Figure 2 Components of the Noscos system

Other components of the system are:

a) *VP:PA FrontEnd (Web)* is a web-based front-end of the system which is used by end-users to interact with the system. It is important to note, that the system implements GUI needed for the end-user (e.g., journalist). Thus, there is no need for end-user GUI in the BalticLSC Instead, the BalticLSC Software should expose its applications as external services.

b) *VP:PA Backend* is the backend of the system. It implements all other functionality of the Noscos system.

c) *Video Task DB* is a database where video processing tasks, their statuses, and results are stored.

d) *Task Listener* is a RabbitMQ-based component that accesses *the Task DB* and communicates the task parameters to the *Video Processing Docker Container*

e) *Video Processing Docker Container* does the actual video processing. It works on a GPU node and takes as parameters a link to a video and a link to the persons database. Since this component is already built as a Docker container, it can be moved to BalticLSC Environment.

In order to implement this user study within BalticLSC Environment:

- The *Video Processing Docker Container* image should be implemented as a BalticLSC module (let's call it *VP:PA module*).
- BalticLSC application should be developed which does a simple task – runs a single computation step using the *VP:PA module*.
- There should be a possibility to start tasks (run applications) and receive the results programmatically, e.g., by calling a web-service from an external system.
- Appropriate changes should be made also in the Noscos system

The user-study is simple in the sense, that only basic features are needed to be implemented in the BalticLSC Environment. It includes also only very basic Computation Application Language features. However, such scenarios require that an end-user possesses strong enough IT-skills in order to integrate BalticLSC within his own system.

2.3.2 Shape optimization for CFD problems

The shape of the boat's hull is of primary importance when it comes to its hydrodynamic properties. The results of the numerical simulation of the phenomenon allow you to determine the distribution of pressures in the water and therefore forces acting on the sailing boat. Such calculations are carried out by CFD (Computational Fluid Dynamics) software.

The project will use the OpenFOAM and FEniCS platforms, which are offering stable and popular solvers for fields problems described by the PDE (Partially Differential Equations). They are both available on an open-source basis, so the author's modifications and adjustment to the BalticLSC requirements are possible.

The CFD simulator will be placed in the Docker container in such a way that it can be run on a wide class of computing nodes. The input to the simulator will be a parametric description of the shape of the object (e.g. boat hull) and its velocity, and the output will be a hydrodynamic force of water resistance.

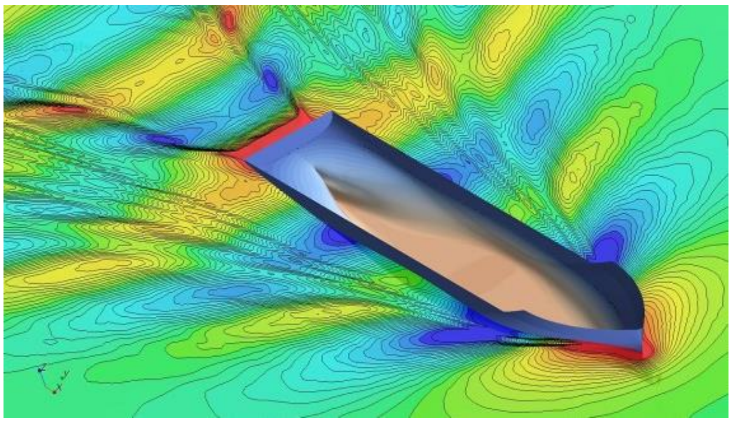


Figure 3 Example of CFD simulation results for marine hull. Distribution of the pressure determines water resistance, boat speed and engine efficiency. (examplary image from Siemens multimedia materials)

Using the simulator module, a system will be created to optimize the shape of the object using evolutionary algorithms. Algorithms of this type are characterized by high computational requirements, but at the same time guarantee an even search of the entire parametric space. Thanks to this, the obtained result is optimal in the global sense.

Evolutionary algorithms are based on multiple, independent calculation of the fit function for subsequent generations of individuals representing the specific shape of the object being optimized. So they are relatively easy to parallelize. BalticLSC application language will be used to control the optimization.

In order to implement this user study within BalticLSC Environment:

- The CFD solver should be developed and placed in the Docker container as a BalticLSC Computational Module called SO.
- Module SO input will be set of the parameters composed of less than 100 numbers with constraints. Its output is also a set of numbers which represent a solution.
- Evolutionary optimization could be developed in two ways:
    - As a Dedicated Computational Module which will control whole optimization process and call the SO Module as an external evaluator for the objective function.
    - The whole logic of the evolutionary optimization will be written directly in the BalticLSC Computation Application Language.

The result of the computation will be a set of parameters for the optimal design, or Pareto front if the problem is in the class of multi-objective optimization.

2.3.3 Pattern recognition in images: recognition of tumor region.

In recent years, deep learning has brought a revolution to the field of pattern analysis and machine learning, by providing algorithms with the capacity to learn complex representations from the raw data itself, achieving human and even super-human level performance in some fields, including medical image analysis.

We developed a pipeline to distinguish four growth patterns of pulmonary adenocarcinoma (LAC) and separate tumor regions from non-tumor. Primary lung adenocarcinomas are heterogeneous tumors that commonly exhibit a mixture of different histologic growth patterns and molecular profiles that are distinct from those of squamous cell carcinoma. The developed method is based on the Convolutional Neural Network (CNN) to automatic detection and segmentation of four lung adenocarcinomas (LAC) patterns on a histological specimen. The user study comes from the Warsaw University of Technology in Poland and is a part of the project: "Development of deep learning methods as a tool to support the analysis of the pathological slides" founded by the National Science Centre, Poland (2016/23/N/ST6/02076).

The developed method allows for automatic evaluation of whole slide images and predominant pattern detection. Steps of method developing and training of a deep learning model were time- and computational-expensive. However, the most computation-intensive step is an analysis of a large cohort of data. This step we would like to delegate to an external service. The reason behind it is the lack of their own computation resources. Currently, a single slide analysis takes 2-4 hours on a PC computer.

Single whole slide image (WIS) includes biological tissue scanned at x40 or x20 (pixel size 0.24 µm or 0.12 µm). The typical size of WSI is in range 2 to 10 GB. During the classification, the whole tissue area is classified on a patch-based fashion, where a single tile is 600x600pixels. Tiles are extracted with 30% overlapping. During the whole slide classification, we can distinguish the following steps:

- open and read a slide,
- extract tiles to classification,
- CNN classification each of tile,
- add classification result to the detection mask,
- save mask.

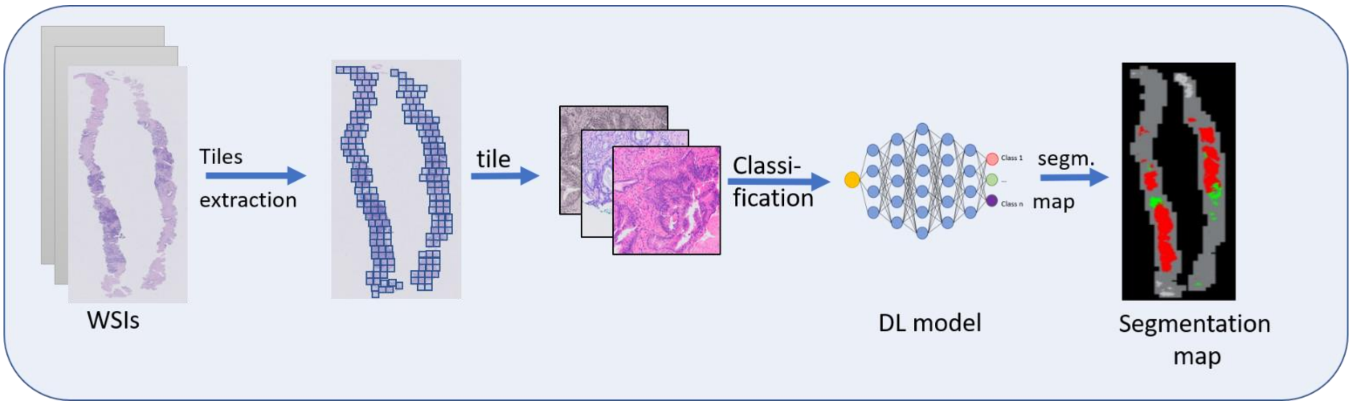


Figure 4 Main steps of the developed method

In order to implement this user study within BalticLSC Environment:

- The slide classification module should be implemented as a BalticLSC module (let's call it SC module). The tiles extraction and classification steps should be paralleled allowing to classify many tiles at the same time.
- BalticLSC application should be developed which does a simple task – runs a computation for a single slide using the SC module.
- There should be a possibility to start tasks (run applications) and receive the results programmatically, e.g. by calling a web-service from an external system.

3. Conceptual model

The conceptual model captures the main concepts of the BalticLSC Environment and relations between them. It is a UML Class diagram (metamodel) of the BalticLSC Software. It describes:

- **The main concepts:** Computation Application, Computation Module, Data Store, and Computation Task.
- **The asset stores:** AppStore and Computation Module Store and Assets they may contain.
- **The user accounts**: Computation Account and the associated software components.
- **The main computation resources**: Master Node and Cluster Node.

The main classes are red in the diagrams. The enumerations (used as classifiers) are green. The grey classes are described in another diagram more detailed.

The conceptual model also defines the main concepts of the Computation Application Language (see Sections 3.1 and 3.2). In fact, it describes the possible abstract syntax of the CAL. We include also an example of the CAL program in the concrete syntax of the CAL in order to illustrate the CAL constructs (see Section 3.3).

3.1 CAL: Computation Applications

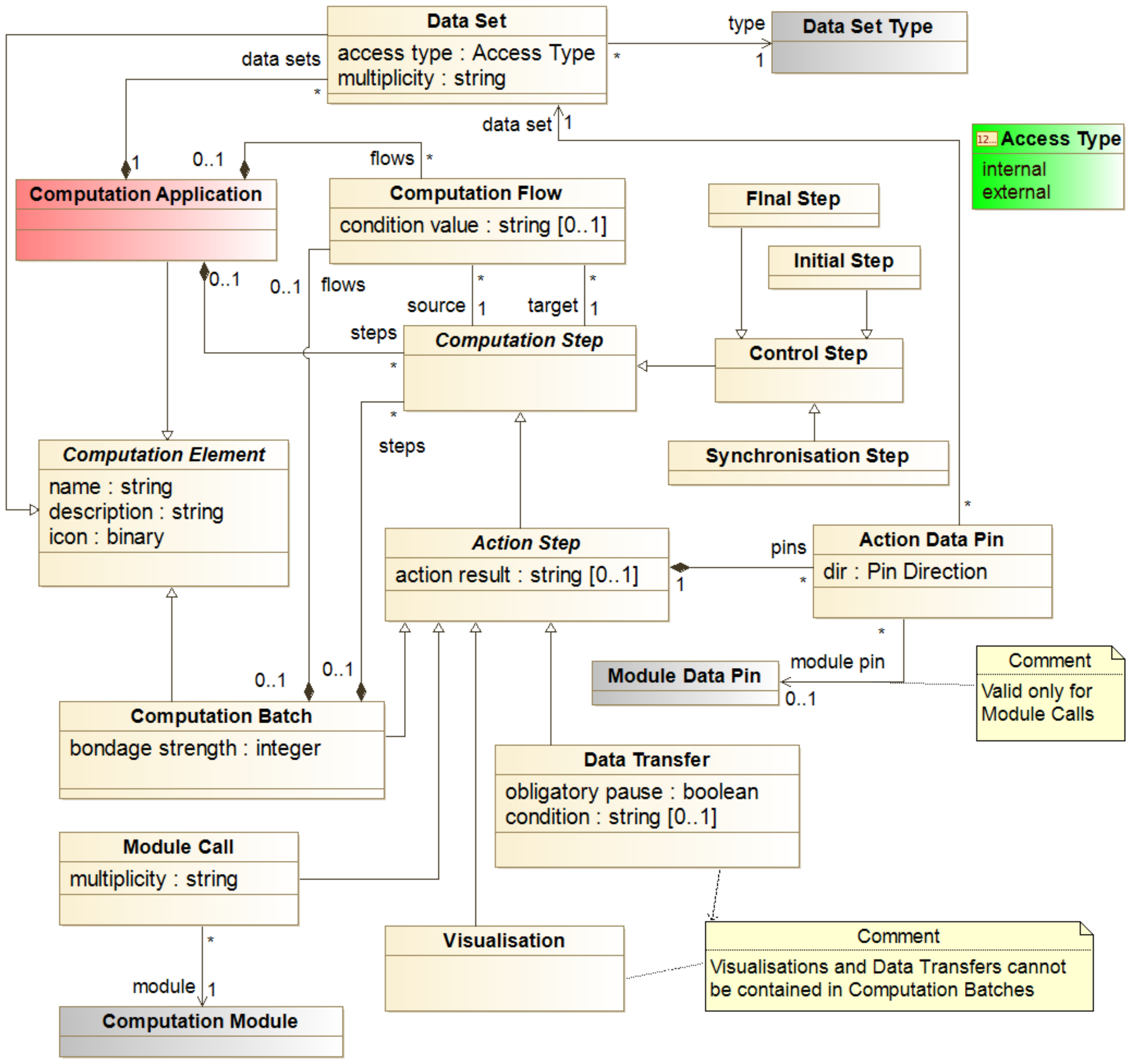


Figure 5. Class Diagram for Computation Applications

Computation Application
A program in the CAL language that can be run by the user. Contains all the Computation Steps and Computation Flows, defining the way computations should be executed. It also contains declarations of Data Sets that will be used as inputs or outputs for the computations.

Computation Step
A single step in the flow of computations. Specializes into concrete steps (see below).

Computation Flow
Specifies flow of computations between Computation Steps. Can contain a condition value that guards flow over the current Computation Flow, based on the result of condition value of the source Computation Flow.

Control Step
A type of Computation Step that does not perform any data processing, but controls flow of execution.

Initial Step, Final Step
Type of control steps that determine the starting and final point of computations. There can be one of each with any Computation Application or Computation Batch.

Synchronisation Step
A type of control step that synchronises parallel computations. Waits for all the action steps on incoming flows to finish, and then runs all the action steps on the outgoing flows in parallel

Action Step
A type of Computation Step the performs data processing and ends with an action results that can be used to control the flow of execution in further processing.

Module Call
A type of action step that executes a specific Computation Module.

Computation Batch
A type of action step that executes a batch, containing specific data processing execution. It contains Computation Steps and Computation Flows. Cannot contain Data Transfers and visualisations. It defines bondage strength that determines assignment of internal Module Calls to Cluster Nodes.

Data Transfer
A type of action that transfers data between some Data Sets. The way the operation is performed depends on the Data Stores associated with the Data Sets. If the input Data Store is marked as “user interaction” then the user receives a signal to access an appropriate data entry form. In other case, data is transferred from some preset storage (file, database, etc.). Data can be transferred to a preset store (internal or external) of a type that matches the input data store.

Visualisation
A type of action that allows to visualize data. Probably not implemented in the early version.

Data Set
A declaration of some data to be used as input or output for Action Steps. Data sets comply with specific Data Set Types, and during runtime should be associated with specific Data Stores.

Action Data Pin
A declaration of some Data Set as input and/or output to the given Action Step.

3.2 CAL: Computation Modules and Data Stores

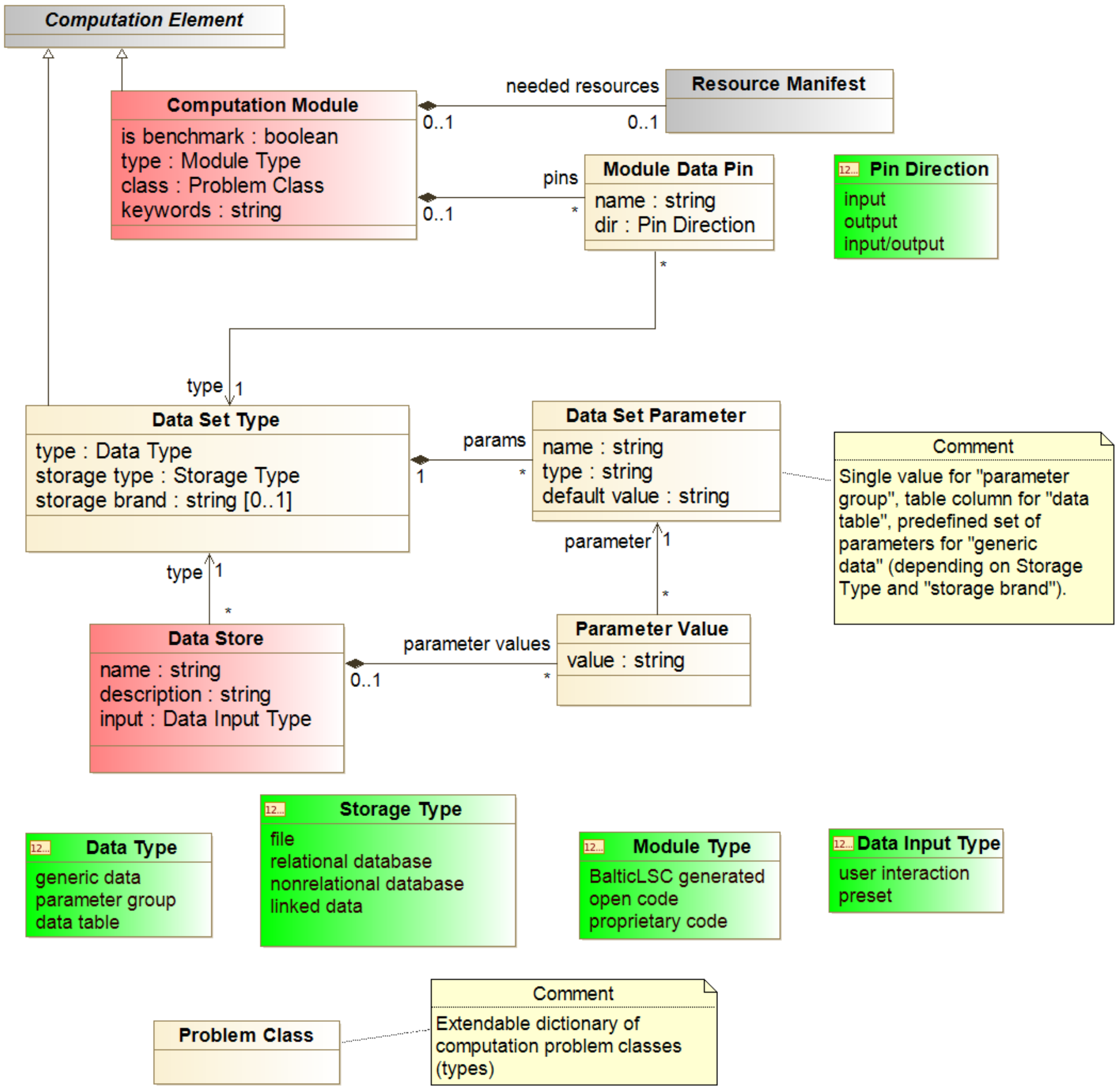


Figure 6. Class Diagram for Computation Modules and Data Sets

Computation Module

A declaration of an executable module that can perform computations according to its internal code. It contains a Resource Manifest that declares resources needed to run the module. It also contains one or more Module Data Pins that declare types of data that need to be input and/or output from the module. Modules can be assigned to a specific Problem Class and keywords that facilitate searching.

Module Data Pin

A declaration of some Data Set Type as input and/or output to the given Computation Module.

Data Store

A declaration of a specific external source/destination for data. It contains a set of parameter values that determine the contents of the Data Store, determined by the associated Data Set Type. Depending on the storage type and data type, these parameter values can contain specific values for parameter groups associated with computations, column names for tables or access parameters for generic data sets.

Data Set Type

A declaration of a predefined type for instantiating various data sets and data stores used in computations. These sets types can be defined and accessed by computation users. They can contain references to data tables, parameter groups or generic data.

Data Set Parameter

A declaration of a single parameter within a Data Set Type.

Parameter Value

A specific value specified within a Data Store. Can be blank if the Data Store is of type "user interaction".

3.3 CAL Example

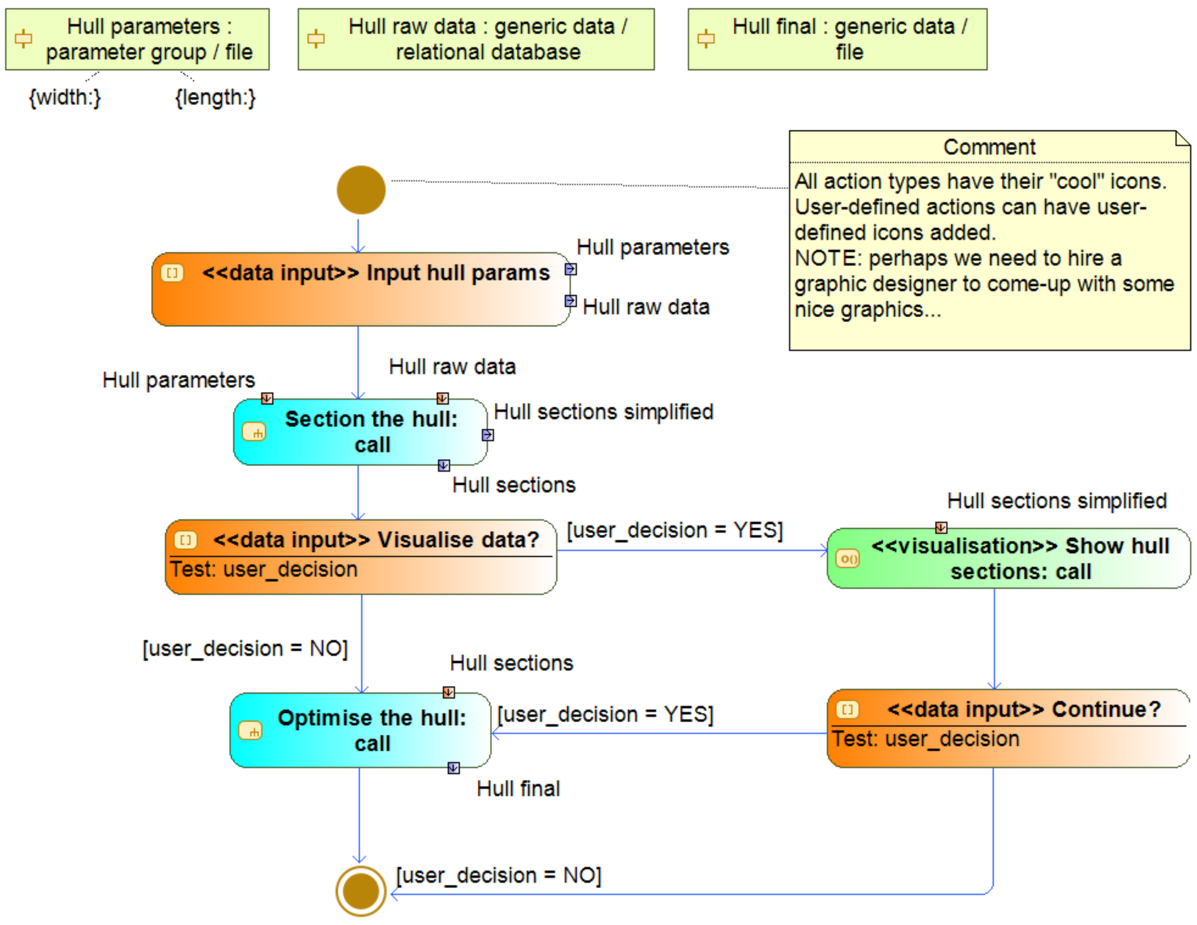

Hull parameters : parameter group / file
Hull raw data : generic data / relational database
Hull final : generic data / file
{width:}
{length:}
Comment
All action types have their "cool" icons. User-defined actions can have user-defined icons added.
NOTE: perhaps we need to hire a graphic designer to come-up with some nice graphics...
<<data input>> Input hull params
Hull parameters
Hull raw data
Hull parameters
Hull raw data
Section the hull: call
Hull sections simplified
Hull sections
<<data input>> Visualise data?
Test: user_decision
[user_decision = YES]
Hull sections simplified
<<visualisation>> Show hull sections: call
[user_decision = NO]
Hull sections
Optimise the hull: call
[user_decision = YES]
<<data input>> Continue?
Test: user_decision
Hull final
[user_decision = NO]

Figure 7. CAL Example – main program

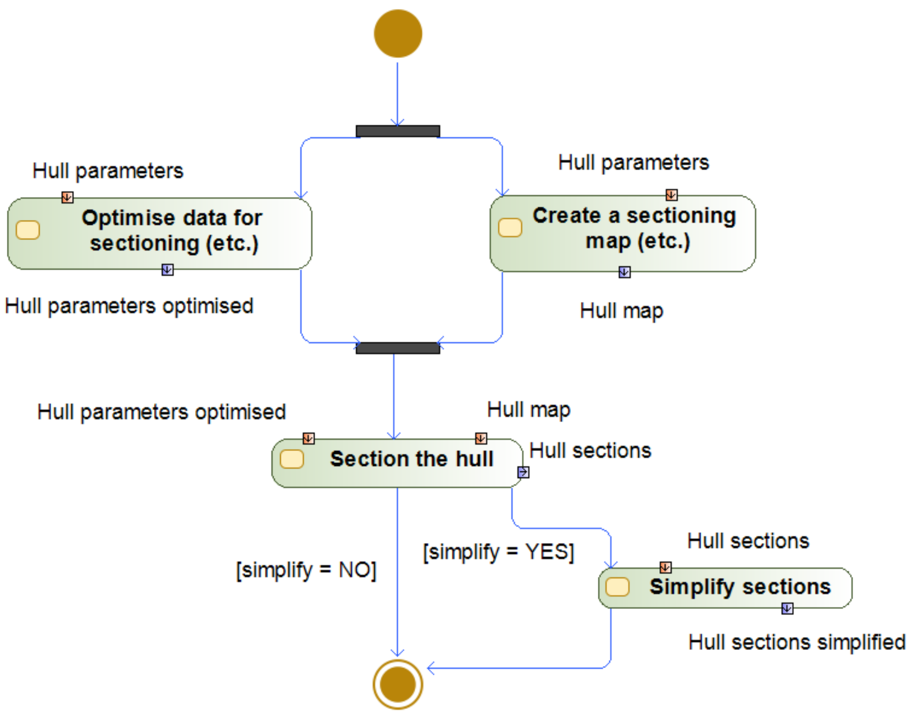
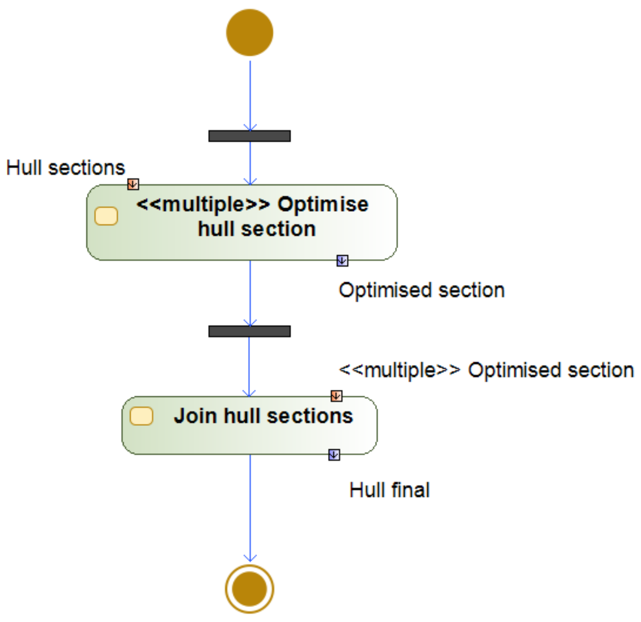


Figure 8. CAL Example – "Section the hull"



Hull sections
<<multiple>> Optimise hull section
Optimised section
<<multiple>> Optimised section
Join hull sections
Hull final

Figure 9. CAL Example – "Optimise the hull"

3.4 Computation Execution and Billing

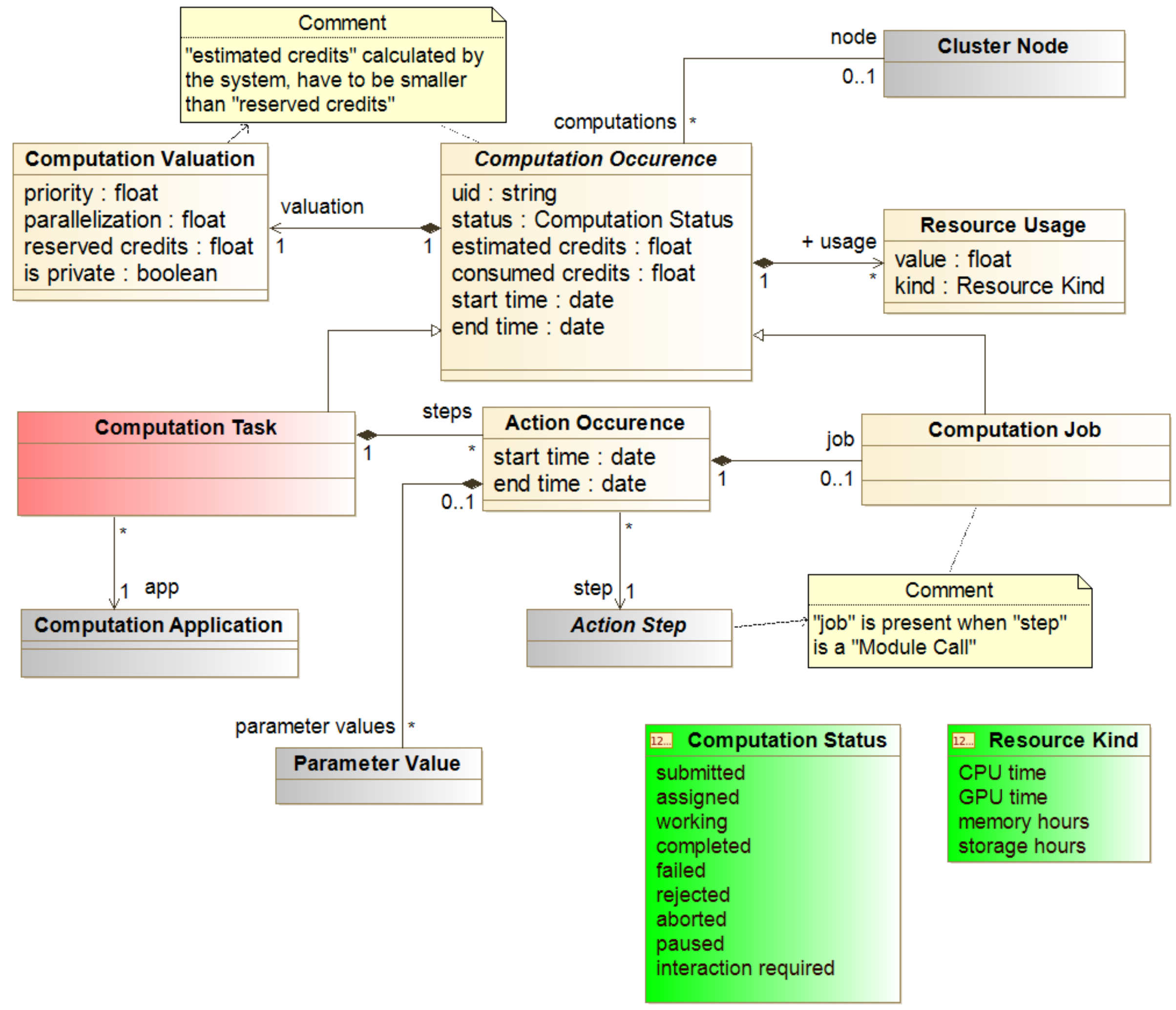


Figure 10. Class Diagram for Computation Execution and Billing

Computation Occurence

A generic type that defines parameters for Computation Tasks and Computation Jobs. Each Computation Occurrence has a Computation Valuation to determine priorities for the given occurrence. It also records its Resource Usage, changes its status according to computation progress and records credit consumption. Normally, each Computation Occurrence is assigned to a specific Cluster Node for its execution.

Computation Task

An instance of a Computation Application and kind of Computation Occurrence. It serves recording all the Action Occurrences associated with the given instance of computations.

Action Occurence

An instance of Action Step within a Computation Application. It records start and end time and specific parameter values for the given action. In case of Action Steps being Module Calls, it also records the Computation Job.

Computation Job

A kind of Computation Occurrence that is associated with a specific Module Call. Computation Jobs are instantiated along execution of Module Calls and assigned to specific Cluster Nodes for execution.

Computation Valuation
A set of parameters that determine priorities for the given Computation Occurrence. Based on these parameters, the occurrences are brokered to nodes of varying computation power.

Resource Usage
A record of usage of a specific resource of a specific Resource Kind.

3.5 Assets and Stores

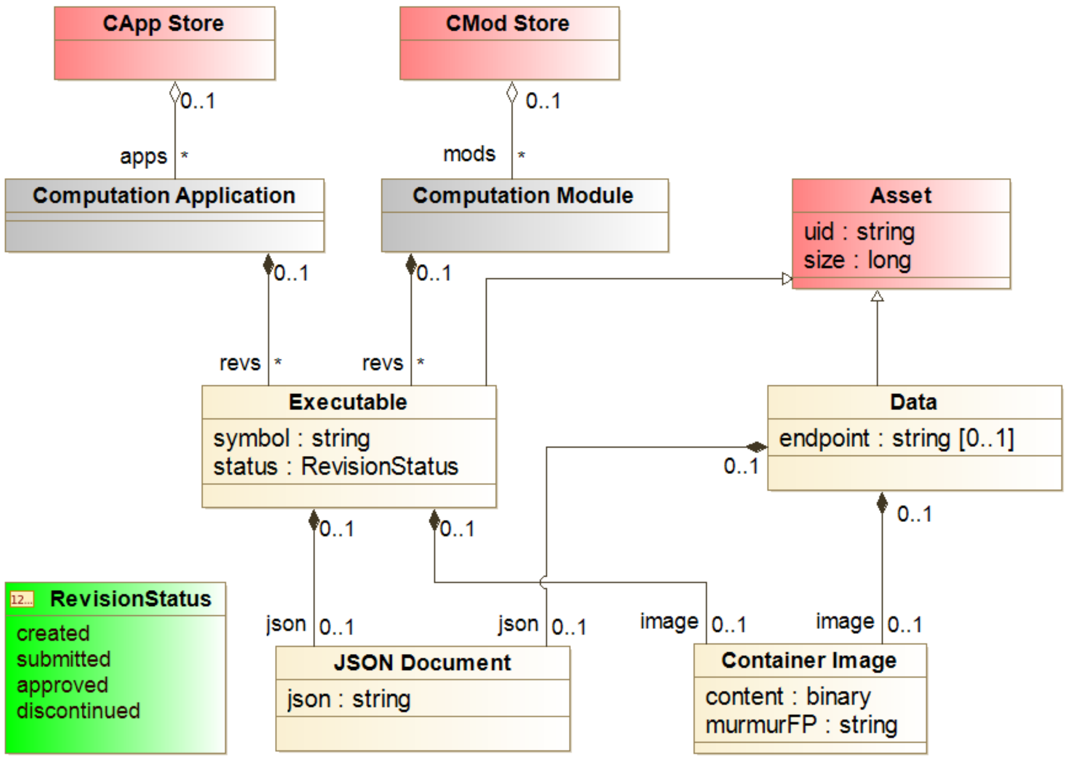


Figure 11. Class Diagram for Assets and Stores

CApp Store
A central (but possibly distributed) store that contains verified Computation Applications that can be used by the users to conduct their computations.

CMod Store
A central (but possibly distributed) store that contains verified Computation Modules that can be used by the users to write their Computation Applications.

Asset
Some data or executable that is stored in the asset repository within the system. It defines its unique ID and size.

Executable
An asset that contains an executable unit for execution within the system. It can be included as part of a Computation Application or a Computation Module. Normally, in the first case, the executable contains a JSON Document, and in the second case – a Container Image. Each executable specifies the symbol for the current revision and its Revision Status.

Data

An asset that contains some data used for processing within the system. If it contains some parameters, then these are held with a JSON Document, if it contains generic data, then it is held in a Container Image.

JSON Document

A file containing a JSON-based specification for some asset.

Container Image

A file containing a container-based content for some asset.

3.6 Computation Accounts

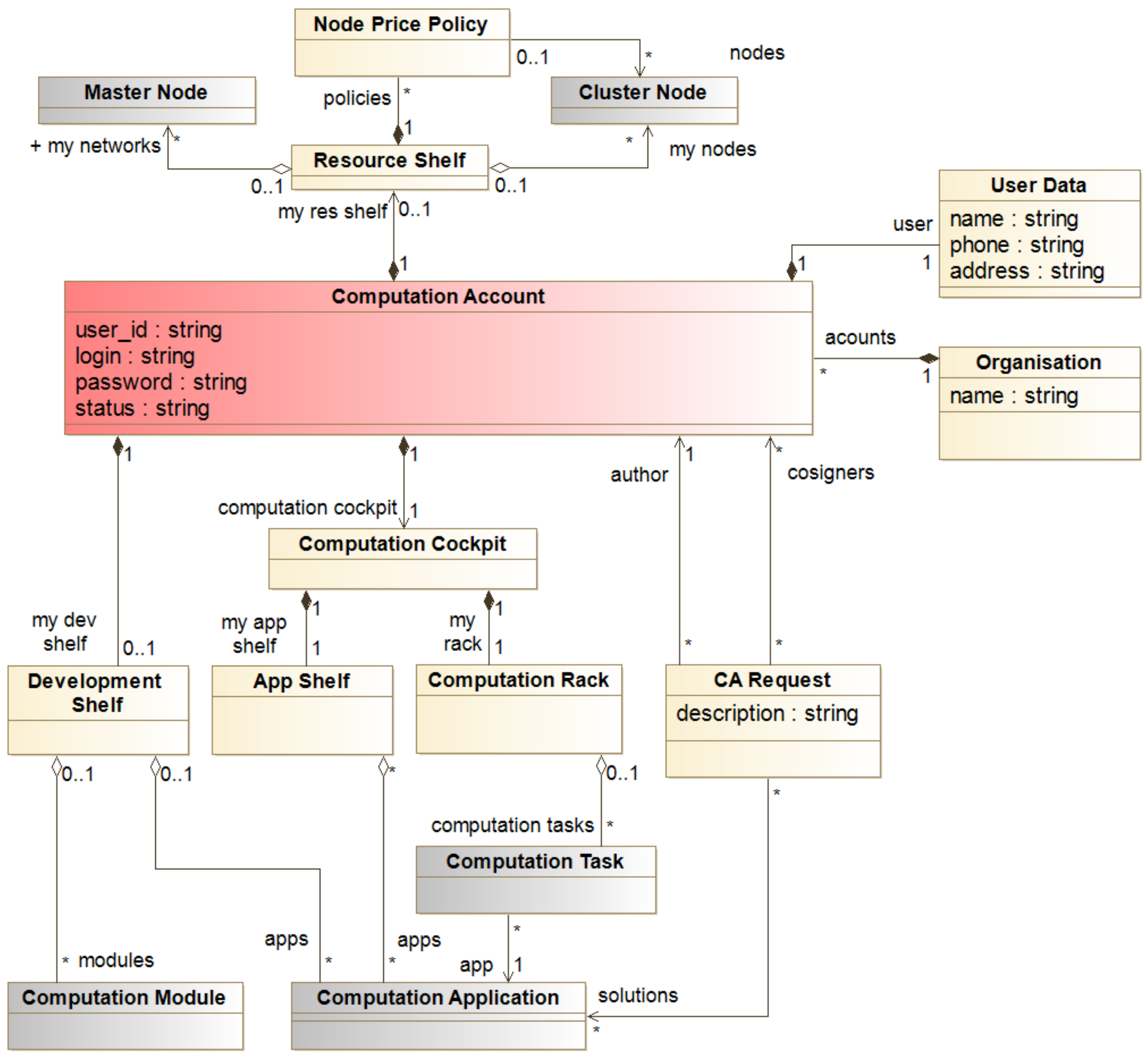


Figure 12. Class Diagram for Computation Accounts

Computation Account

A record for a specific user of BalticLSC. It gathers all the information about the user and determines the user's access rights.

Resource Shelf

Gathers all the information about Nodes owned and managed by the given Computation Account, including Node Price Policies (structure of the policies not certain at this point – no business models available).

Computation Cockpit

Gathers all the information regarding computations performed by the given Computation Account. This includes the App Shelf that gathers all the Computation Applications chosen the user and Computation Rack that gathers all the Computation Tasks run by the user.

Development Shelf

Gathers all the information regarding development activities for the given Computation Account. This includes the developed Computation Applications and Computation Modules.

CA Request

A request to develop/update some specific application.

3.7 Computation Resources

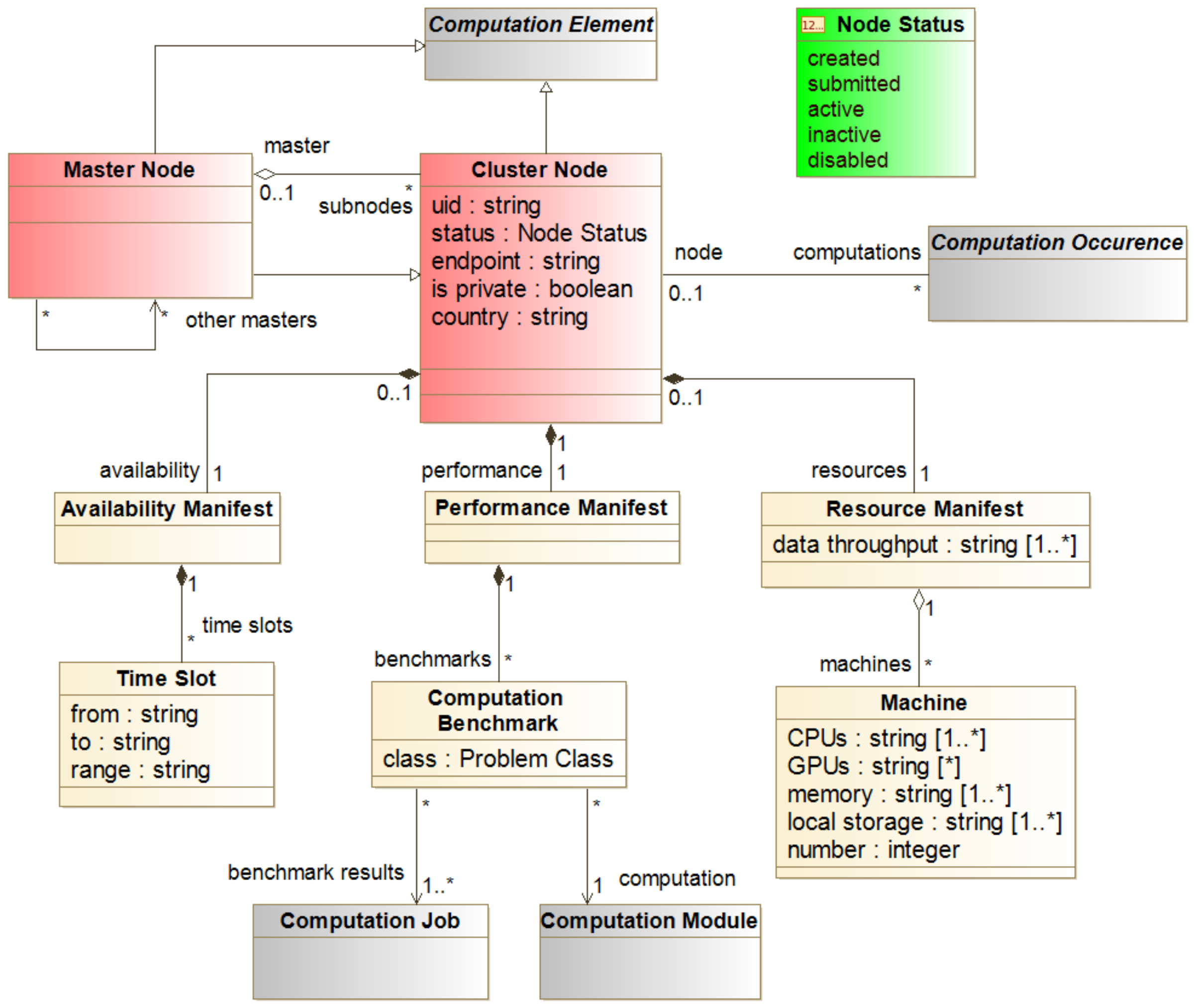


Figure 13. Class Diagram for Computation Resources

Cluster Node

A register entry with information about a specific Cluster Node (a set of machines organized into a computation cluster). The Cluster Node gathers information on availability, performance and resources (machines) declared for the described cluster. Do not confuse it with Kubernetes Node – it is a Machine in the model! Cluster Node corresponds to the Kubernetes Cluster.

Master Node

A distinguished Cluster Node that gathers several subnodes and communicates with other Master Nodes to organize the network.

Availability Manifest

A description of available Time Slots for a given Cluster Node. The Time Slots perhaps need to have more information than just "from-to" (e.g. what if a running job exceeds the "to"?).

Performance Manifest

A set of Computation Benchmark results defining the computation power of a given Computation Node.

Resource Manifest

A set of machine declarations, available as part of the given Cluster Node.

Computation Benchmark

Measurement of the performance of the predefined computation on a computation resource in order to ensure the conformation to the manifest, e.g., Kubernetes does not provide detailed information about the type of CPU on a node. Although it provides information on the number of CPU cores, it is unknown how many CPUs the node has and it is unknown if hyper-threading is enabled. Because cost of CPUs will be set the same across the whole cluster, it can be of interest to have some benchmarking results from each node. This information could be used during job/workload scheduling.

4. Functional Requirements

This chapter describes the functional requirements of BalticLSC Software. First, we describe the most important functionality of the BalticLSC Software using a UML Use-case model (see Section 4.1). Next, the functionality to be implemented in the first sprints of the development is detailed using User Stories (see Section 4.2).

4.1 Use-case model

This section describes the most important use-cases (functions) of the BalticLSC Software. We divide the functions in several groups:

- Computation Application Usage (End-user functions).
- Computation Application Development (Developer functions).
- Computation Resource Management (Supplier functions).
- Computation Resource Supervision (Administrator functions).
- Computation Application Requests (End-user-Developer interaction functions).

We have omitted here "obvious" functionality which is required by a modern web-based system with user-authentication, e.g., login, logoff, user management. We have not defined precisely also functionality related to the business model of the BalticLSC. This includes the method of how to determine the price of the calculation and the procedure of signing the user agreement and steps if this agreement has been violated, etc. … The mentioned functionality would be described later.

4.1.1 Computation Application Usage

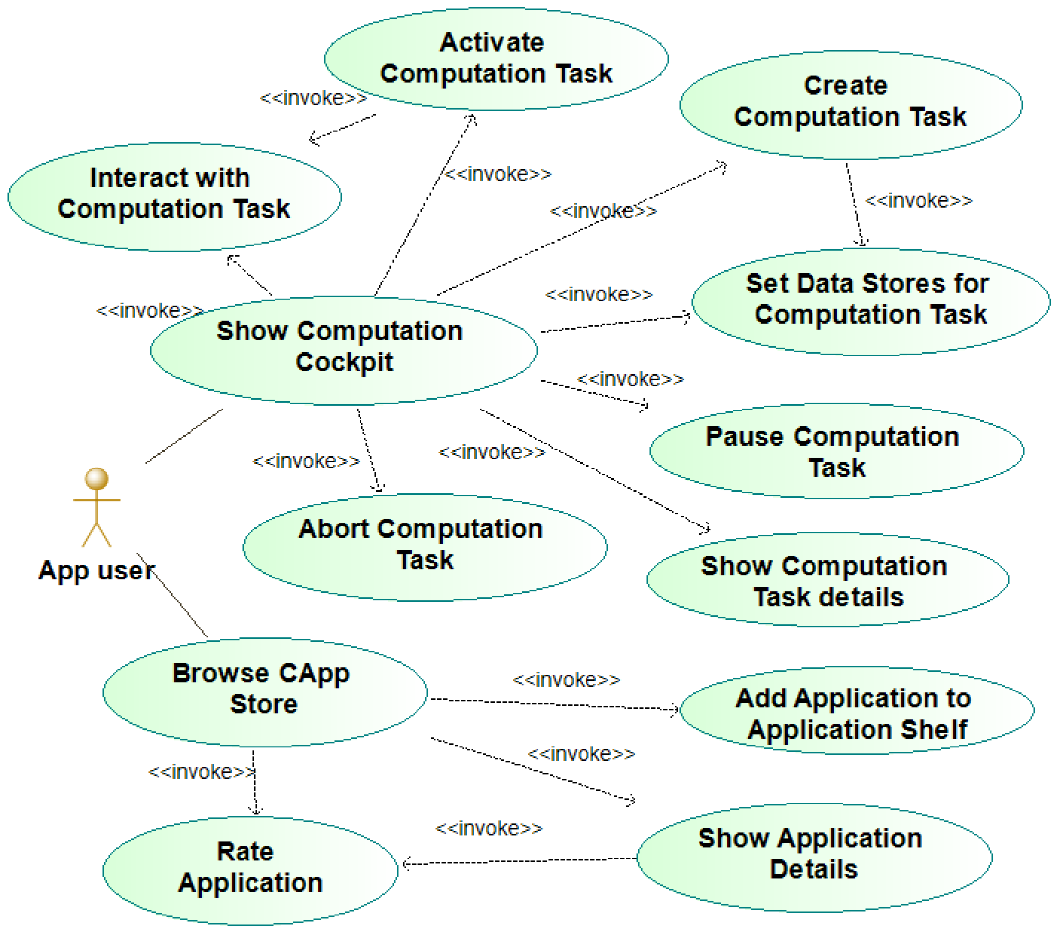


Figure 14. Use Case Diagram for Computation Application Usage

Name	Show Computation Cockpit
Description	Show the App Shelf with all the Computation Applications that the user can run – retrieved from the CApp Store or created by him/her. Also shows the Computation Rack with all the Computation Tasks created by the user, based on those applications.

Name	Create Computation Task
Description	Prepare to run (but not run yet) a new Computation Task using a selected Computation Application. Includes specifying the Computation Valuation.

Name	Activate Computation Task
Description	Start executing an existing Computation Task which is just created or has been paused. Might involve interacting with the task.

Name	Interact with Computation Task
Description	Performs required interactions (entering parameters, selecting options, etc.) with a selected Computation Task. Available only for the Computation Tasks that need such interactions.

Name	Set Data Stores for Computation Task
Description	Allows to define parameter values for Data Stores as specified by the Computation Application. This use case can be used dynamically during computations to change e.g. ways in which the Computation Application interacts with the App user (pauses for data input etc.).

Name	Show Computation Task details
Description	Show details of a Computation Task including its valuation and configuration. If task execution was started, shows also the task status and statuses of its subtasks.

Name	Abort Computation Task
Description	Stops execution of a running Computation Task. All the computations are aborted and cannot be activated in the future. Appropriate billing for the computations performed so far is calculated.

Name	Pause Computation Task
Description	Stops execution of a running Computation Task. All the computations are halted but can be activated in the future. All the internal data is stored and billing for storage is calculated.

Name	Browse CApp Store
Description	Allows for browsing of verified Computation Applications in the CApp (Computation Application) Store.

Name	Add Application to App Shelf

Description	Add a selected Computation Application to the user's App Shelf, which makes it available in the Computation Cockpit.

Name	Show Application Details
Description	Show detailed information about a Computation Application.

Name	Rate Application
Description	Allows the user to review a selected Computation Application.

4.1.2 Computation Application Development

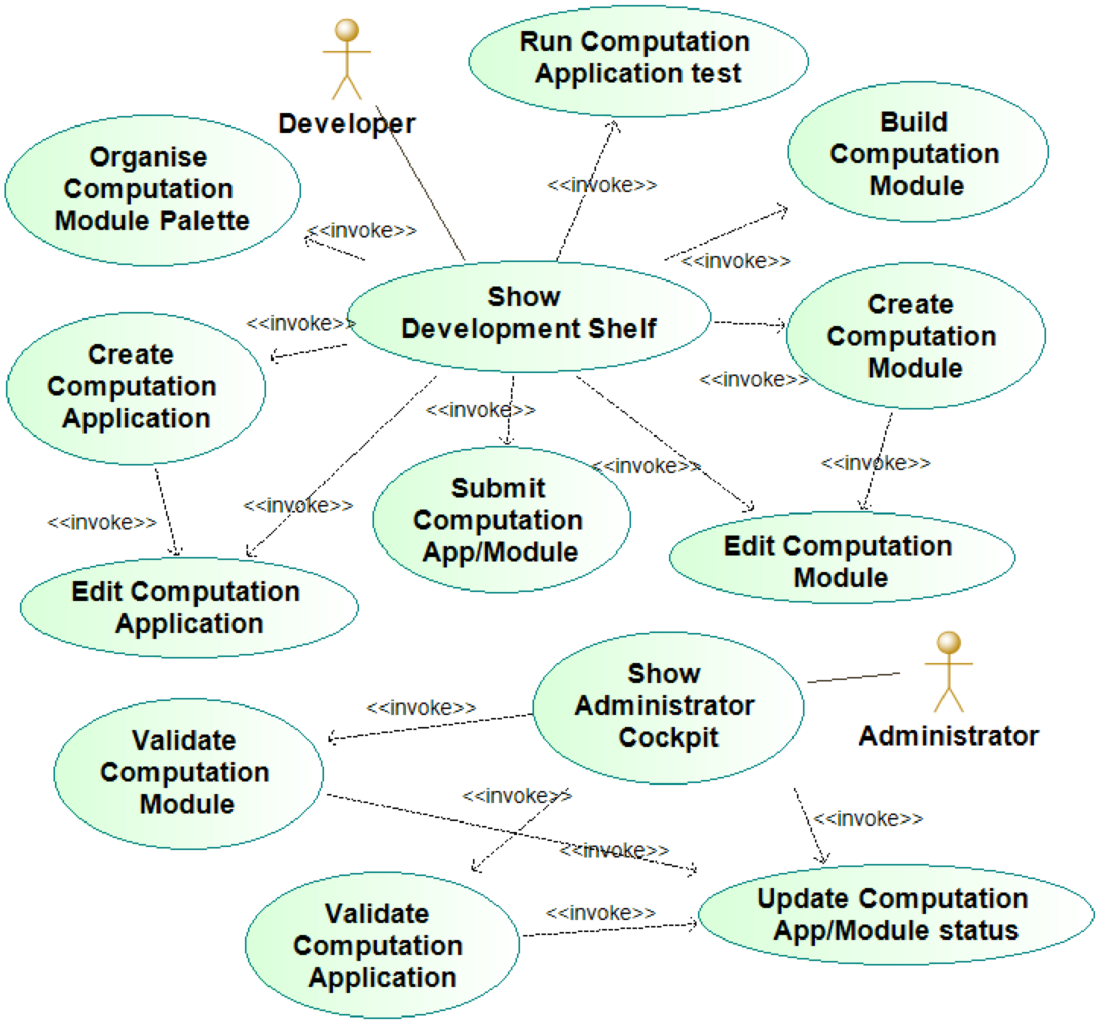

Developer
Run Computation Application test
Build Computation Module
Organise Computation Module Palette
<<invoke>>
Show Development Shelf
Create Computation Module
Create Computation Application
Submit Computation App/Module
Edit Computation Module
Edit Computation Application
Show Administrator Cockpit
Administrator
Validate Computation Module
Validate Computation Application
Update Computation App/Module status

Figure 15. *Use Case Diagram for Computation Application Development*

Name	Show Development Shelf
Description	Allows to browse all the Computation Modules and Applications developed by the current user.

Name	Submit Computation App/Module
Description	Allows for submitting a Computation Module or Application to be verified and made available to other users.

Name	Create Computation Application
Description	Create a new Computation Application entry.

Name	Edit Computation Application
Description	Specify content of a Computation Application using the CAL editor.

Name	Create Computation Module
Description	Create a new Computation Module entry.

Name	Edit Computation Module
Description	Specify content of a Computation Module – its description and executables.

Name	Build Computation Module
Description	Prepare the executable form of a computation module (meant for future versions).

Name	Organise Computation Module Palette
Description	Select Computation Modules to be available in the Palette for the given Developer. Allows also to organize order of Modules in the Palette.

Name	Run Computation Application test
Description	Allows for running of developed Applications in restricted test mode on the BalticLSC Network.

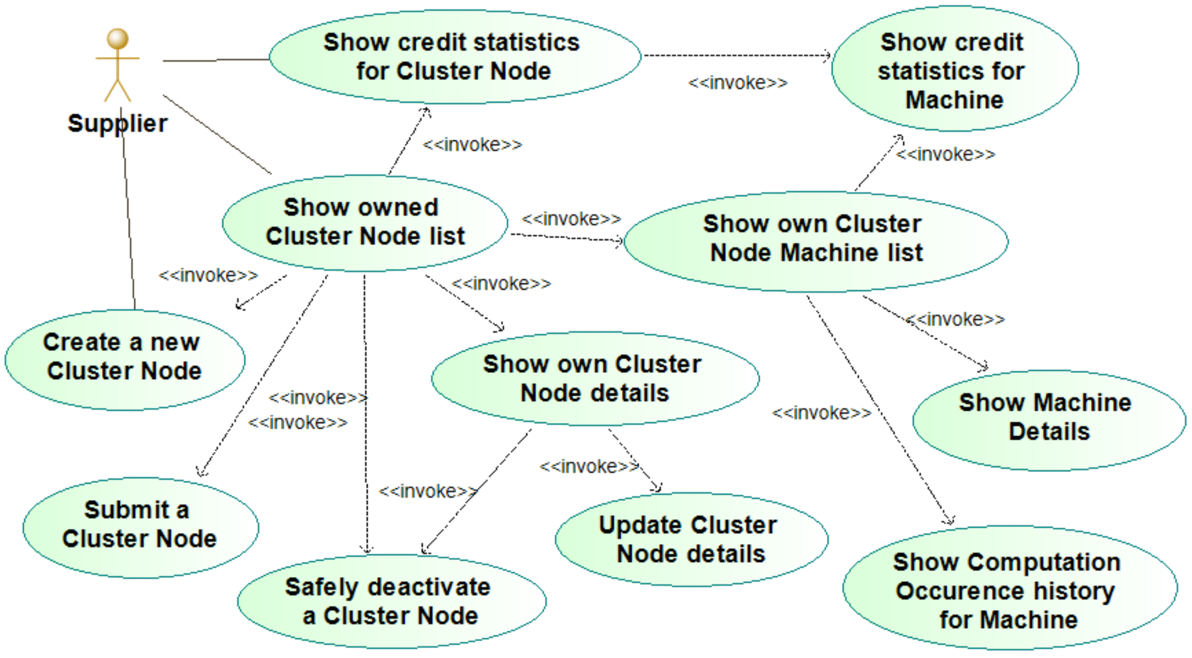
Name	Show Administrator Cockpit
Description	Shows the App/Module Repository which allows to browse all Computation Modules and Applications that have been ever submitted by any user (including active and inactive).

Name	Validate Computation Application
Description	Show advanced details about a Computation Application, including its implementation (code) and allow to validate its usage in the BalticLSC Network.

Name	Validate Computation Module
Description	Show advanced details about a Computation Module, including its implementation (code) and allow to validate its usage in the BalticLSC Network.

Name	Update Computation App/Module Status
Description	Allows for status change, including approval of a Computation Application or Module.

4.1.3 Computation Resource Management



Supplier
Show credit statistics for Cluster Node
Show credit statistics for Machine
Show owned Cluster Node list
Show own Cluster Node Machine list
Create a new Cluster Node
Show own Cluster Node details
Show Machine Details
Submit a Cluster Node
Update Cluster Node details
Safely deactivate a Cluster Node
Show Computation Occurence history for Machine
<<invoke>>

Figure 16. *Use Case Diagram for Computation Resource Management*

Name	Create a new Cluster Node
Description	Define a new Cluster Node entry.

Name	Show owned Cluster Node list
Description	Show all Cluster Nodes defined by the current user.

Name	Submit a Cluster Node
Description	Submit a Cluster Node for verification, after which it will be made available to other users for performing computations.

Name	Show own Cluster Node Machine list
Description	Show all Machines for a selected Cluster Node.

Name	Show credit statistics for Cluster Node
Description	Show statistic about credits earned by the user’s Cluster Node.

Name	Show credit statistics for Machine
Description	Show statistic about credits earned by the user’s Machine.

Name	Show own Cluster Node details
Description	Show details about a selected Cluster Node.

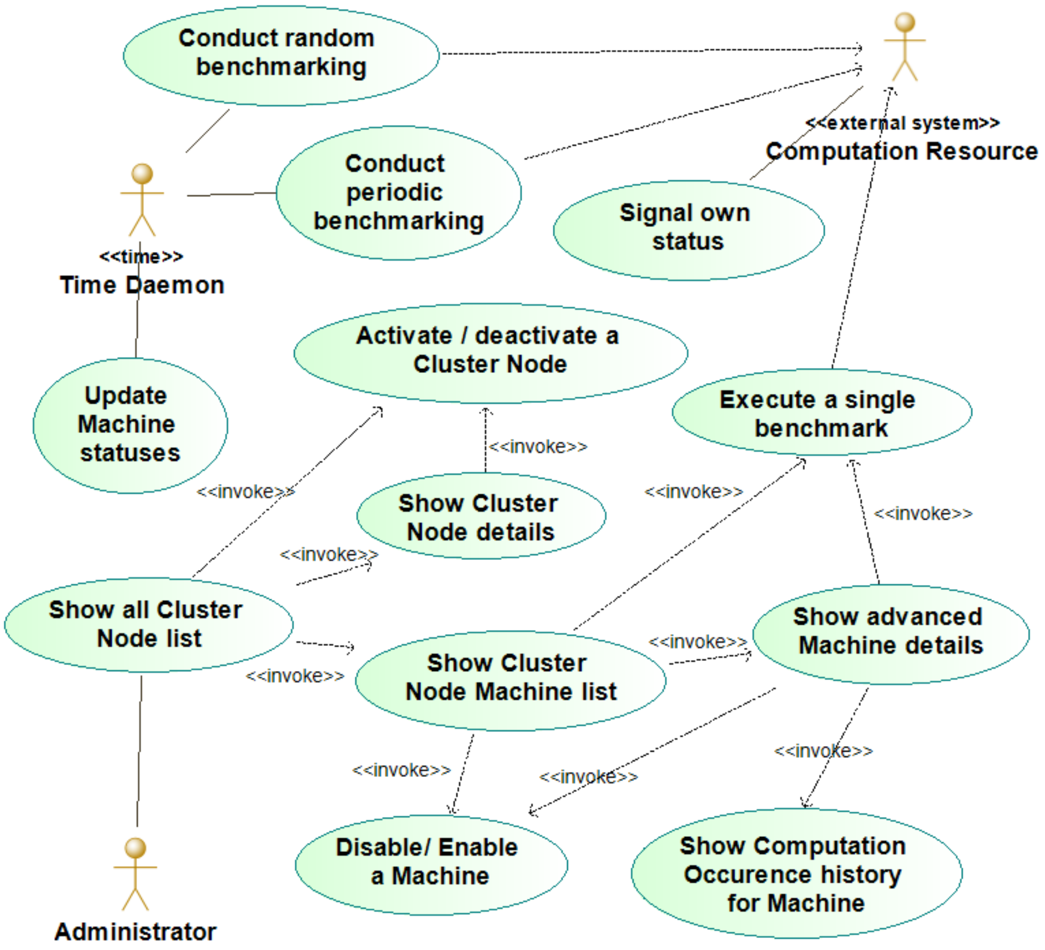
Name	Show Machine details
Description	Show details about a selected Machine.

Name	Show Computation Occurrences history for a Machine
Description	Show information about all the Computation Occurrences that have been ever executed on a selected Machine. Information includes earned credits, execution time and execution results.

Name	Safely deactivate a Cluster Node
Description	Stop receiving new Computation Tasks by a Cluster Node and deactivate it after all already assigned Computation Tasks are completed. Optionally: freeze and save all Computation Task and deactivate Cluster Node right away.

Name	Update Cluster Node details
Description	Allows for modifying information that describes a selected Cluster Node.

4.1.4 Computation Resource Supervision



Conduct random benchmarking
<<external system>>
Computation Resource
Conduct periodic benchmarking
Signal own status
<<time>>
Time Daemon
Activate / deactivate a Cluster Node
Update Machine statuses
Execute a single benchmark
<<invoke>>
Show Cluster Node details
Show all Cluster Node list
Show Cluster Node Machine list
Show advanced Machine details
Disable/ Enable a Machine
Show Computation Occurence history for Machine
Administrator

Figure 17. Use Case Diagram for Computation Resource Supervision

Name	Show all Cluster Node list

Description	Shows a list of all submitted Cluster Nodes.

Name	Show Cluster Node Machine list
Description	Shows a list of machines for a selected Cluster Node.

Name	Show Cluster Node details
Description	Show all details about a Cluster Node.

Name	Activate / deactivate a Cluster Node
Description	Allows to make a selected Cluster Node available or unavailable for performing computations by the users other than its owner.

Name	Show advanced Machine details
Description	Shows all details about a Machine, including its benchmark results.

Name	Disable / Enable a Machine
Description	Allows to make a selected Machine available or unavailable for performing computations of users other than its owner.

Name	Execute single benchmark
Description	Choose and run a benchmark on a selected Machine.

Name	Conduct random benchmarking
Description	Run benchmarks on randomly selected set of Machines.

Name	Conduct periodic benchmarking
Description	Periodically run benchmarks on all Machines.

Name	Signal own status
Description	Allows a Cluster Node to inform the BalticLSC Network of its machines’ statuses and send information about progress of Computation Occurrences assigned to them.

Name	Update Machine statuses
Description	Mark Machines that didn’t inform about its statuses as inactive.

4.1.5 Computation Application Requests

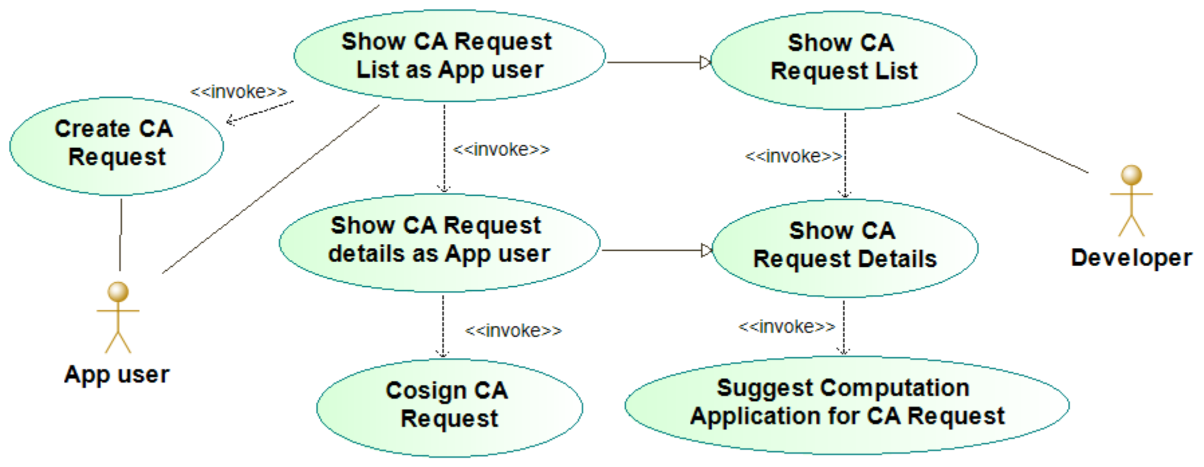


Figure 18. Use Case Diagram for Computation Application Requests

Name	Create CA Request
Description	Allows to create a Computation Application Request specifying details of a Computation Application user needs.

Name	Show CA Request List/Show CA Request List as End-user
Description	Show a list of all created Computation Application Requests.

Name	Show CA Request Details/Show CA Request Details as End-user
Description	Show the definition of a selected Computation Application Request.

Name	Cosign CA Request
Description	Mark the current user as being interested in a Computation Application Request created by another user.

Name	Suggest Computation Application for CA Request
Description	Allows to assign computation applications as fulfilling a selected Computation Application Request.

4.2 User stories

This Section contains user stories that describe the use-cases in detail. User stories capture the possible functionality of the use-case (or use-case slice), data needed to input and show in the UI, and also to the open issues currently raised. Thus, they are not the final requirement specification and are the subject of changes, unless stated otherwise explicitly.

The list of the user stories is currently limited to the use-case slices for the ongoing design and development sprints[1]. Each user story is organized as follows:

- User story – who performs the function, what is his intention, and why he does it.
- Name of the user story.

[1] The content of the BalticLSC Software development sprints is available in the project's GitLab system - https://www.balticlsc.eu/gitlab/baltic_lsc/product-backlog/-/boards for authorized users only.

- Name of the related use-case (or use-case slice).
- Acceptance criteria – the expected behaviour of the software.
- Open issues to be discussed.

User Story:	As an *end-user* I want to be able to view, sort and filter (browse) the available applications so I know what apps I can be run in the BalticLSC Environment
Name:	**Browse Available CAL Applications**
Related use case:	"Browse CApp Store" - allows for browsing of verified Computation Applications
Acceptance criteria:	1) Shows all *Computation Applications* with status *approved*. 2) For each available app following info should be shown: • application name • short description • application author • icon • date (datetime) when last updated • app rating • tags • categories • mark if the app is in the user's App Shelf 3) It should be possible order and filter (search) applications.
Open issues:	1) What type of ordering should be used? E.g., order by last updated, popularity, similarity to my interests ... 2) What type of filtering should be used? E.g., filter by organization, author, … 3) What other dates are important? E.g., creation date, approval date, … 4) What other popularity/quality measures can be used? E.g., the number of executions, views, … 5) One possible idea for UI: https://hub.docker.com/search?q=&type=image

User Story:	As an *end-user* I want to be able to add a specific application from a list of available CAL applications to my app shelf so I can run it in the BalticLSC Environment
Name:	**Add Application to App Shelf**
Related use case:	"Add Application to App Shelf" adds a selected Computation Application to the user's App Shelf, which makes it available in the Computation Cockpit.
Acceptance criteria:	1) An end-user is able to select an app (or several apps) from a list of available CAL applications. 2) The end-user can add the selected app (or several apps) to his App Shelf. 3) After performed addition, the end-user can see the added CAL application(s) in his App Shelf.
Open issues:	1) Should App Store and App Shelf be visible simultaneously?

	2) Is there any agreement to be ratified before adding an app to the App Shelf? Possible UI solutions: 3) A list with available CAL applications is shown. After clicking on an item of this list, the item (CAL Application) is marked as added to user`s App Shelf (no multi-select, action on every click) 4) A list with available CAL applications is shown. Every item of this list contains a checkbox showing if this app is going to be added to the user's App Shelf. User can add an app to his App Shelf by setting the checkbox state to checked/unchecked and pressing the button (multi-select, action on a special command)

User Story:	As an *end-user* I want to be able to view (browse) CAL applications present on my App Shelf so I know what apps I can run in the BalticLSC Environment
Name:	**View Apps in My App Shelf**
Related use case:	"Show Computation Cockpit " shows the App Shelf with all the Computation Applications that the user can run – retrieved from the CApp Store or created by him/her
Acceptance criteria:	1) Shows all *Computation Applications* present in the user`s App Shelf. 2) For each available app the following info should be shown: • application name • short description • application author • icon • date when added to the shelf
Open issues:	1) Should it be possible to choose the order? 2) Should some kinds of filters be present? One possible idea for UI: https://hub.docker.com/search?q=&type=image

User Story:	As an *end-user* I want to create a computation task so that I can run computation application
Name:	**Create a Computation Task**
Related use case:	"Create Computation Task" prepares to run (but not run yet) a new Computation Task using a selected Computation Application. Includes specifying the Computation Valuation. **Note!** We describe a simplified use-case – a very basic application with just one step and no parameters.
Acceptance criteria:	1) A new task corresponding to the selected Computation Application is created and added to a list of user`s tasks. 2) Computation valuation of the newly created task gets default values: ● priority - Medium ● parallelization - No

	• is_private - No 3) The end-user is asked to provide reserved credits (ComputationValuation.reserved_credits). 4) The newly created task gets the initial status
Open issues:	1) Should come up with a better name for *Valuation*, e.g., Properties or Priorities. 2) What is the default status of a newly created task? 3) Explain semantics of: • parallelization • is_private 4) Define default values for priority, parallelization, is_private.

User Story:	As an *end-user* I want to start created computation task so that I obtain my computation result
Name:	**Start Computation Task**
Related use case:	"Activate computation task" starts executing an existing Computation Task which is just created or has been paused. **Note!** We describe a simplified use-case – a very basic computation task with just one step and no parameters.
Acceptance criteria:	1) A job is created from a module call present in a computation application and matched to a node in a computation cluster for further execution. 2) Job's execution is started on the specific node in the resource cluster. It is started if the estimated amount of credits does not exceed the reserved amount. 3) Job's start- and end-time are recorded. 4) During the execution of the job information on resource usage is collected and stored. 5) Info on resource usage is added to cluster node resource usage statistics. 6) Job's execution is aborted if the amount of really consumed credits exceeds the number of reserved credits specified in computation valuation.
Open issues:	1) Where do the computation occurrence's estimated credits come from? How it is computed? Is it computed after the job has been matched to a node? 2) Specify what information on resource usage is stored.

User Story:	As an *end-user* I want to view tasks created by me so that I monitor the execution status of my tasks
Name:	**View User's Task List**
Related use case:	"Show Computation Cockpit" shows Computation Rack which contains all user's tasks.
Acceptance criteria:	1) The user can see a list of tasks created by him. 2) For every task the following info is provided: • task_uid

	• link to computation application • status • start_time • end_time (if already completed) • already consumed_credits • reserved credits • overall progress
Open issues:	1) Should the task list be sortable, filterable? 2) Is it possible to show overall progress in percentage terms?

User Story:	As an *end-user* I want to view jobs started by me so that I monitor the execution details of my tasks
Name:	**View User's Jobs**
Related use case:	"Show Computation Cockpit" shows Computation Rack which contains all user's jobs.
Acceptance criteria:	1) A list of jobs started by this user is shown. 2) Every item in this list contains: • job uid • readable name • reference to the parent task • start time • end time (if finished) • job status • consumed credits • reserved credits • progress
Open issues:	1) Is there a readable name for a job present in the model? 2) Is computation valuation set for task as a full or for every job (module call) separately? E.g., credit reservation? 3) Can there be a repeated module call in one task? How to set computation valuation separately in this case? Can there be cycles in computation apps? 4) Are sorting and filtering applied here?

User Story:	As an *end-user* I want to view task details so that I monitor the execution details of the specific task
Name:	**View Task Details**
Related use case:	"Show Computation Task details" show details of a Computation Task including its valuation and configuration. If task execution was started, shows also the task status and statuses of its sub-tasks.

	Note! We describe a simplified use-case – a very basic computation task with just one step and no parameters.
Acceptance criteria:	1) Following information should be present: • task_uid • link to computation application • status • start_time • end_time (if already completed) • already consumed_credits • reserved credits • priority • parallelization • is_private • overall progress • detailed resource consumption • description of node assigned for execution 2) Info on subtasks (now we have just one subtask), so we can just show one corresponding item from the jobs list (see user story View User's Jobs).
Open issues:	1) What details should be shown for resource consumption? 2) What details should be shown for the assigned node?

User Story:	As an *end-user* I want to view job details so that that I monitor the execution details of the specific job
Name:	**View Job Details**
Related use case:	"Show Computation Task details" show details of a Computation Task including its valuation and configuration. If task execution was started, shows also the task status and statuses of its subtasks.
Acceptance criteria:	1) Following info should be present: • job uid • readable name • reference to the parent task • start time • end time (if finished) • job status • consumed credits • reserved credits • progress 2) Detailed resource consumption should be present: • GPU time • CPU time • memory hours • storage hours
Open issues:	1) Is there a readable name for a job present in the model? 2) How memory and storage hours are computed?

5. Non-functional requirements

Short name	**Integration with External Systems**
Id	WUT_NF1
Description	It is important that the user can use the environment he already knows and which provides other functionality (not computation-intensive). Therefore, the BalticLSC Software should be able to accept orders from other applica-tions via plug-ins. Integration with external systems through a BalticLSC API or plug-in mechanism.

Short name	**“Fire and Forget”**
Id	WUT_NF2
Description	“Fire and forget” – the end-users prefer to start an app and forget about the rest until the computation results are available. Therefore, the BalticLSC Software should be able to execute computations without interfering with an end-user as few times as possible.

Short name	**Computation Packages**
Id	WUT_NF3
Description	Grouping of Computation Modules into “Computation Packages” – such packages must be executed on a single Computation Node. BalticLSC Soft-ware should allow to identify such modules and to assure their execution on the same Computation Node.

Short name	**Distance from Data**
Id	WUT_NF4
Description	BalticLSC Software should be able to take into the account the distance (“Strength” of network connection) between Data Source where data resides and the Computation Node where the computation occurs. The computation should take place as close as possible to the data source. The same principle should be used when parallelization is done by distrib-uting computations over separate physical nodes – the optimization of over-head for data transmission should be made.

Short name	**Crash Handling**
Id	WUT_NF5
Description	The BalticLSC Software should handle Computational Module crashes/freezing. Errors should be expected situations.

Short name	**Action Logging**
Id	WUT_NF6
Description	The BalticLSC Software should perform action logging for debugging, bill-ing, code profiling, fraud detection purposes.

Short name	**Computational Gauging**
Id	WUT_NF7
Description	The BalticLSC Software should provide evaluation of computation resources (CPU, GPU, RAM, etc.) using a common evaluation unit. It should enable both standard and custom “exchange rate” for various resources.

Short name	**Computational Market Exchange**

Id	WUT_NF8
Description	The BalticLSC Software should solve billing or market trading problems. Advertisement exchange market mechanisms could be used for calculating optimal pricing for computations (computations are performed on nodes that offer best value for money). Note, that Pekao SA expertise could be used.

Short name	**Computation security**
Id	IMCS_F5
Description	The BalticLSC Software should assure the privacy of the input and output data and of the computation itself. The access to the data should be granted to the computation executor, but access to the computation execution metadata should be granted also to the resource owner in the context of the resource usage.

Short name	**Computation reliability**
Id	IMCS_F6
Description	The BalticLSC Software should assure the availability of the provided resources while computation executes. System should assure that the provided resources perform accordingly to their specification.

Short name	**Easy access**
Id	LIC_NF1
Description	The BalticLSC Software should be available online using ordinary web-browser. It should have a single point of contact and a convenient user interface.

Short name	**Simple definition of the computation application**
Id	RQ_NF1
Description	The BalticLSC Software should offer as simple means of application definition as possible. Average end-user should be able to understand the basic control and data flow of the application. Learning to develop the computation application should not take lots of time and efforts (e.g., one lecture).

6. Conclusion

The main objective of the document is to describe the user requirements for BalticLSC Software. The user requirements specification is a “live” document and it is updated during the process of software design and development. The main work is the addition of the necessary details to the user requirements, e.g., by providing more user scenarios according to the work plan of the agile design and development process.

This document is the main source of the information for the ongoing activities A5.2 Design of the BalticLSC Admin Tool, A5.3 Design of the Computation Application Language, and A5.4 Design of the BalticLSC Computation Tool. This document is also used in the activity A6.2 Implementation of the BalticLSC Software in order to ensure the match between user requirements and developed software.