Grant Agreement no: 690770

Ship Lifecycle Software Solutions (SHIPLYS)

Project Deliverable Report

D3.3 Requirements for the integration of SHIPLYS tools and compatibility with existing tools

Version:	2.5
Author:	(BMT)
Contributors:	Thomas Koch (AES), Gary Randall (BMT), Rhyan Hoey (BMT),
Internal reviewers:	José Ignacio Zanón Millán (Soermar), Ujjwal Bharadwaj (TWI), Paul Brown (TWI)
Deliverable due date:	2017-05-31
Actual submission date:	2017-09-19
Work package:	WP3
Task:	T3.3
Dissemination level:	Public
Lead beneficiary:	BMT
Status:	Final

VERSIONS

Version	Date	Reason	Editor
1.0	2017-01-17	First release to internal contributors	F de Castillo
1.1	2017-07-17	New Table of Contents	G Randall (BMT)
2.2	2017-09-08	Release to internal review	G Randall (BMT)
2.3	2017-09-13	Internal review	J.I.Zanon (SOERMAR)
2.4	2019-09-15	Internal review	U Bharadwaj; Paul Brown (TWI)
2.5	2017-09-18	Finalised	Randall/Hoey (BMT)

Acknowledgement:

The research leading to these results has received funding from the European Union's Horizon 2020 research programme under grant agreement No. 690770.

EXECUTIVE SUMMARY

The SHIPLYS project focuses on developing and integrating rapid virtual prototyping tools with life cycle tools. This is in order to empower European SME designers and production yards to make their own decisions on how to optimise early stage activities via a life-cycle approach, both in the realm of optimum ship design and when optimising retrofitting and conversion activities.

This report outlines the requirements needed to ensure software compatibility of any new SHIPLYS tools with the existing early design tools. It defines a variety of integration methods and a programming interface that supports interaction between disparate software components. Our over-arching requirement is that any component to be considered for inclusion in SHIPLYS must be capable of implementing one of these methods or at least interacting with another component that acts as a go-between. Such components are embedded in an architecture that defines services that operate both between them and at the global SHIPLYS level. These service types include meta-data, data, software registry and job services. Many associated factors have to be considered, including (a)synchronicity of execution, access authorisation and time to complete a given job.

The report further provides a first sketch of a user interface and a dashboard that allows a designer to track their workflow through an activity map as they interact with various tools coupled to the SHIPLYS platform.

Finally, we include a template for a Software Integration Register in which we record all individual components and any detail about them pertinent to their ability to be integrated. A further update to D3.3 will define which particular components are to be used in order to address the needs of the three SHIPLYS usage scenarios.

CONTENTS

List of figures

Abbreviations

API – Application Programming Interface
BIM – Building Information Modelling
CAD – Computer Assisted Design
CoG – Centre of Gravity
DAI – Data Access Interface
DoW – Description of Work
FEM – Finite Element Modelling
GUI – Graphical User Interface
HATEOAS - Hypermedia As The Engine Of Application State
HCI – Human-Computer Interface
HTTP – Hypertext Transfer Protocol
IRI – Internationalized Resource Identifier
JSON – JavaScript Object Notation
LCC – Life Cycle Cost
LCCA – Life Cycle Cost Analysis
REST – Representational State Transfer
RSET – Rapid Ship Evolution Tool
RVP – Rapid Virtual Prototyping
TTL – Time-To-Live
URL – Uniform Resource Locator
URI - Uniform Resource Identifier

1 Introduction

This report will describe the integration requirements for the tools both developed within SHIPLYS and also interfacing requirements for existing tools to interface with SHIPLYS software. For instance, these integrated tools may need to be compatible with existing early design tools, such as FORAN and we will determine how and what data to transfer data between such CAD systems and production planning systems.

In particular, we will provide key specifications/constraints for the development of the tools in WP5 and WP6, in order to facilitate their integration in WP7.

These requirements will not be presented as a formal list of specific needs, instead we describe the overall ethos that that will allow a variety of tools to interact. That said, there are several desired outcomes that we might expect the final SHIPLYS toolset to support, such as; the integration of multiple RVP and LCCA tools; support for LCCA tools that deal with varying time windows; the ability to take inputs from physically distributed sources; the ability to express risk in a variety of formats; and the need to support intuitive GUI wherever possible.

The SHIPLYS project is technologically positioned at TRL 6/7 in most aspects; the various subsystems – data formats, life cycle cost assessments (LCCA) models, virtual prototyping models – have all been developed and demonstrated at least in 'relevant environments' for their original intended purposes. The thrust of SHIPLYS is mainly the integration of methodologies and bespoken data forms and life cycle performance outputs to the needs of the marine asset industry in a way that facilitates the uptake of SHIPLYS results by SME stakeholders.

Note taken from the proposal: *"The prospective use of "free" software tools could be used in a positive way to underline the openness and a potential way forward to lower entry cost for SME yards even further. However, our proposal to use open source or free software does not mean to switch to this as the main basis. It is important to keep at least one or two established and widely used commercial products at hand, for instance FORAN from our advisory partner SENER. We should also be careful to distinguish between free (close source) tools and open source tools. The former ones are often a problem, as the provider will not be prepared to modify the product without serious business options but the latter may cause substantial effort to get them adjusted to the project's needs. Nevertheless, our experience with open source products has been more positive in general."* (Source SHIPLYS proposal)

2 Approach

2.1 Overview

2.1.1 SHIPLYS Integration Technology

2.1.1.1 Background

The solution to be developed in the SHIPLYS project will – on the software side – be dependent on integration of various software components such as full-scale applications, analytical programs, smaller or larger utilities, database management systems and other data sources, and software libraries. A significant portion of these components is expected or known to have existed before the project commenced and is provided by third parties or partners, while others will be developed by partners as the project proceeds.

In general terms, integration of such components must be based on the following assumptions:

- In many cases, modification of an existing component is either not possible or impractical. This is particularly true for commercial third party components which may often appear in the form of a black-box type of software with no access to sources and limited configuration options. Such components will often mandate a specific runtime environment, which cannot be changed. Consequently, integration should be possible without any real modification of such components.

- Components will often operate on some input and output data streams of known formats combined with interactive control by the user, e.g. applications performing some sort of heavy-weight calculations. It must be assumed that sufficient documentation or similar explanatory material will be available such that some “glue” code can be provided in the project for such components to provide the required input and control data or consume the output of interest.

- Alternatively, components may have their own data management capabilities. For this kind of component some existing data access interfaces should be available.

- A further variant may be constituted by some degree of built-in extensibility e.g. by means of scripting or plug-ins. This is particularly relevant in highly interactive components such as modelling systems.

- It should be possible to develop new components with minimal constraints concerning the implementation technology, e.g. free choice of runtime platform and programming environment and other technical constraints. This is mandated also by the fact that such components will often be based or depend on existing algorithms or sub-components and/or reliance on existing know-how among development staff involved.

2.1.1.2 Design Goals

The following general design goals can be established for the project:

- Minimised programming technology constraints for software components:
    - Runtime platform.
    - Programming platform.
    - Language and time zone.
- Support for distributed operations: components may reside on hosting systems if they are reachable via network.
- Light-weight integration of black-box type components.
- Maximised support of functional requirements of SHIPLYS.

2.1.1.3 Data Model Definitions

One of the most important elements of integration is agreement on the data entities to be communicated between participating software components. This is one of the core concepts defined in the DoW.

Working with a common data model appears as an attractive and elegant solution. However, based on experience from precursory projects, some consideration must be given to the following issues:

- Creation of an appropriate data standard: While many data standards and de-facto standards exist, these may not cover all of the relevant engineering and commercial aspects at the required level of granularity. However, deriving a new fully integrated model is a significant undertaking and is not within the scope of the project. Therefore, an approach based on an amalgamation of various contributing standards and rules, such as that used in Building Information Modelling (BIM), may be the best approach.
- Project development dynamics: Given that multiple work packages and tasks occur concurrently within the project, modifications and corrections to the data model are likely to be needed as the work packages are completed. In order to mitigate this risk, any dependencies of the data model on the progress and outcomes of other tasks should be minimised.

These considerations indicate the need for self-explanatory, machine readable data model definitions that may evolve over time. This means that an implementation should not only provide the classical information management functions for storing, retrieving or locating instances of data entities but should also provide meta data about the underlying model definitions. This is a well-known concept found in various programming environments as well as in some data management systems with different levels of functionality.

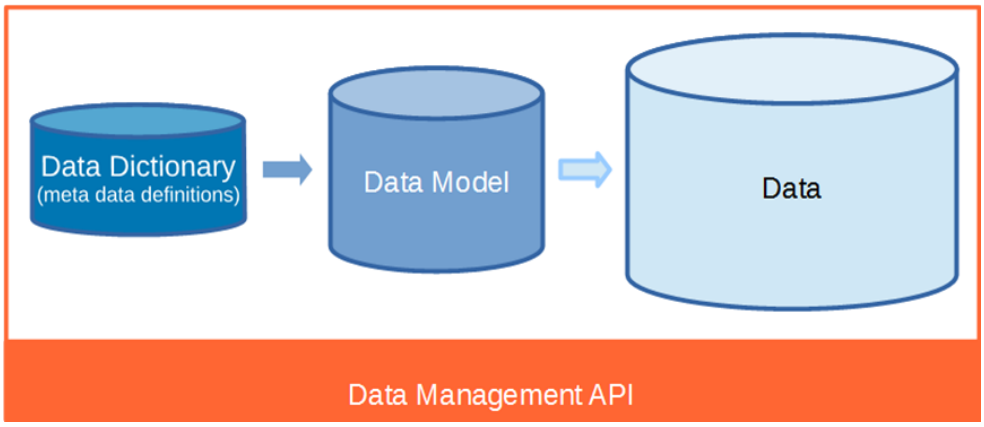

Figure 1: Data Model structure

2.1.1.4 Integration Methods

Glue code

This integration method provides functionality to expose component functionality of black-box type software via a SHIPLYS compliant REST API. This is particularly useful for non-interactive calculation modules.

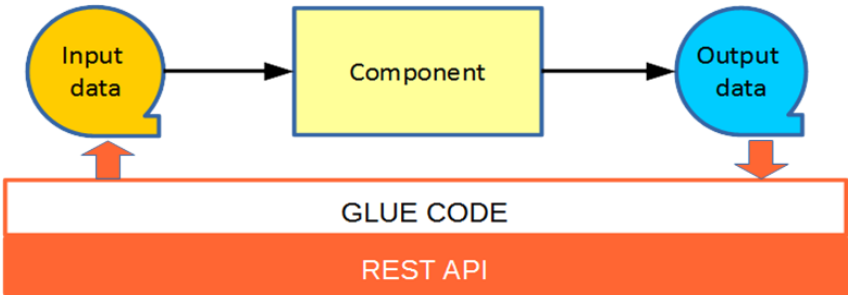

Figure 2: Glue code integration structure

Examples: Standalone module for Weight or CoG calculations, resistance prediction, FEM processor

Data access interfaces

This is applicable to systems operating on some sort of data management platform. It can range from simple file system level retrieval to full-blown data management system access.

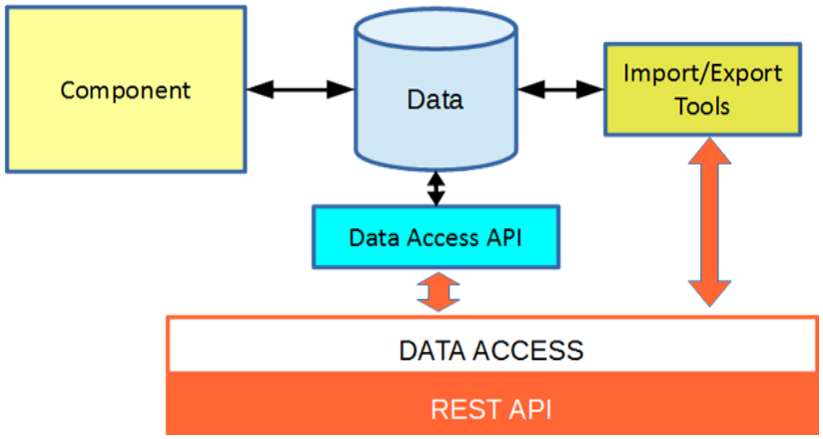

Figure 3: Data access interface integration structure

Examples: AVEVA Marine Database retrieval, AutoCAD DXF import, export or drawing extraction, …

Plug-ins

Some systems or components will provide mechanisms to extend or tailor their functionality such as a plug-in API or a scripting environment. This can be used to enable such systems to interact themselves with the SHIPLYS platform. For example, this may be realised as menu/user interface additions in user interfaces that effect some data retrieval or trigger a calculation in another remote component.

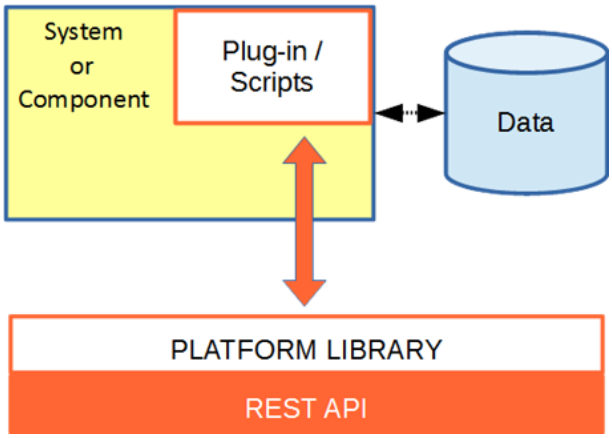

Figure 4: Plug-in integration structure

Examples: most CAD systems provide similar mechanisms, e.g. AutoCAD, FORAN, AVEVA Marine etc.

2.1.1.5 Proposed Solution Ingredients

REST services – all data management and other server components will be accessible via a REST HTTP API utilising JSON encoded payloads. REST over HTTP relies on conventions layered on top of the ubiquitous HTTP protocol. The primary reason for using this approach is the flexibility in implementation both in terms of actual design of the API itself as well as the relative easiness of support across a wide range of implementation platforms and operating systems.

Data model support – in order to accomplish the required self-documenting data model services and to facilitate self-adjusting software components, a meta data interface as part of the REST API will be needed. This interface will provide access to (sub-)schemes, entities, attributes and build-in low-level type definitions. By providing such interface functionality, consuming software will be able to establish references to its local data models.

Data state – to provide data life-cycle support functions, it shall be possible to accompany any entity instance with data state information: version (e.g. as a time stamp), origin (identifying the creating component), quality (e.g. vacant [needed but missing], estimated, calculated, validated etc.).

Function registration – the integration platform will have to provide – through the REST API - a lookup and locator function for software components and the functionality supported by them. Components can register themselves in terms of component identification and description as well as individual operating instances (e.g. CAD system X running on host Y, offering a set of functions Z). A first idea of such functionality may include:

- Register function (component, name, version, description, required input data, offered output data).
- Deregister function.
- Find function (by component, name, description, input or output data).
- Register instance (host, URL).
- Deregister instance.
- Lookup active instance for function.

Example:

https://<server>/find-function/description=weight|centre%20of%20gravity

2.2 Activity model

In order to facilitate the development of the SHIPLYS tool, an activity model has been composed based on the ship design elements of ISO 10303, the Standard for the Exchange of Product model data (STEP). This standard allows for the communication between different modules/tools that will be integrated to be explicitly resolved. Some additions have been made to account for the life cycle analysis functions to be covered within SHIPLYS.

The activity model essentially consists of a sequence of activities involved in the ship life cycle, with connections showing data flows for inputs, outputs, controls and mechanisms associated with each activity node. The top-level overview of the activity model is shown in Figure 5 below. A more detailed introduction to the activity model concept can be found in D3.1, and a full description of the SHIPLYS activity model is given in D3.2.

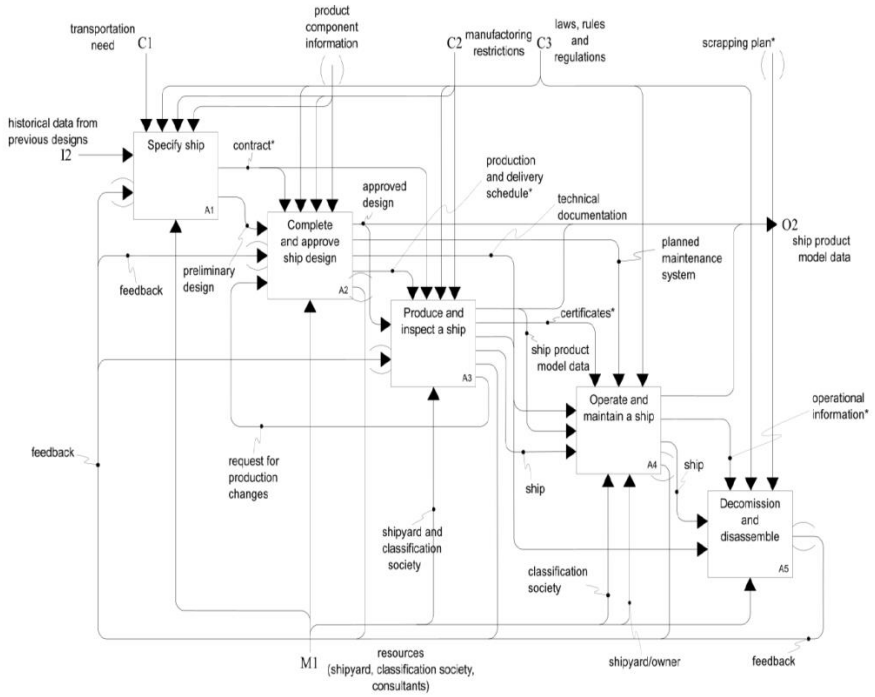

Figure 5: SHIPLYS Activity Model overview

The activity model can serve as a guide for integration of the various software components to be utilised by the SHIPLYS platform, as it maps the various functions covered by the components and illustrates the input, control and output data flows between them. The functionality, inputs and outputs of each software tool can be used to determine which functions within the activity model the tool facilitates. The activity model can then be used to guide integration and highlight any gaps or overlap in functionality of the software tools.

2.3 What might the SHIPLYS tool look like?

Final design of the SHIPLYS offering is still under consideration but there are several broad categories that are candidate services. For instance, SHIPLYS might offer remote access to a platform that is natively running several integrated tools of interest. Or, we might offer a standalone executable that interfaces to the tools already owned by the user. A third possibility is a cloud service that acts as a gateway/exchange between differing relevant tools. There are of course more possible architectures than this, and the choice will depend on commercial viability (especially with regard to licensing), desirability and the real-world usability of the product. Whatever the choice, we can imagine a dashboard-like front end that guides a user through the early design process and allows engagement with LCC and risk assessment tools at appropriate junctures.

A mock-up of such a dashboard is presented here and takes inspiration from the activity model just outlined in section 2.2. This mock-up is part of a larger design effort that is presented in full in section 2.7 of this report.

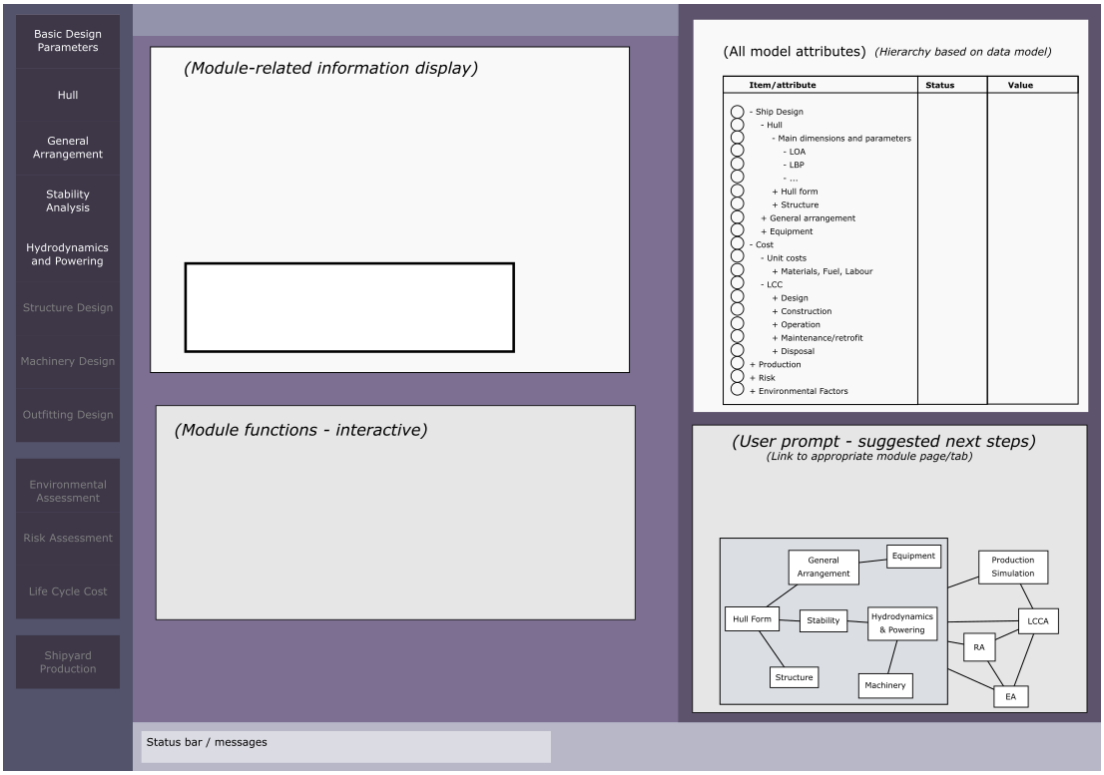

Figure 6: SHIPLYS mock dashboard interface

This blank initial screen shows the project workspace before a project has been initiated. The actual tools and workflow for the interface are yet to be decided. The overall workflow is divided into modules, with sub-functions within each module; these are roughly based on the first and second levels of the Activity Model respectively.

Modules are selected using the buttons on the left-hand side of the interface, while tabs are used to access different sub-functions within each module. These tabs are displayed on the central pane of the interface, with

information and data presented in the top portion and user actions (e.g. buttons to launch other tools) in the lower portion. The right-hand pane provides feedback to the user: the upper panel indicates the status and current values of various attributes of the ship design in an expandable tree, while the lower one shows an interactive graph indicating workflow progress.

The following section outlines the REST API that underpins the component interaction shown in Fig. 6.

2.4 REST Based Web Service: Functional Requirements

This chapter describes the functional requirements for the REST API that will be used to enable different software parts/components within the SHIPLYS framework to communicate and exchange data via three main service components we define. Section 2.5 illustrates possible Use Cases for the functional requirements.

Figure 7 shows the sequence of main events and activities regarding the usage of the REST API within the SHIPLYS framework. The figure shows only part of the sequence and does not include the execution of the tool(s) that will be carried out.

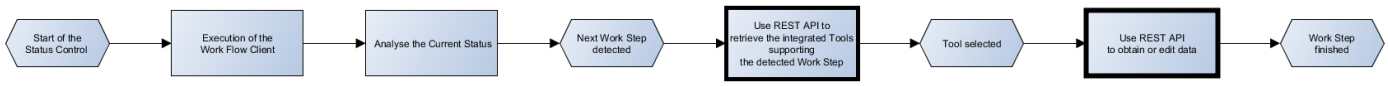

Figure 7: Principal interaction with API

2.4.1 General requirements

2.4.1.1 HTTP Methods and return codes

The HTTP-methods GET, POST and DELETE (in very rare occasions) shall be usable with their implied semantics and the server shall return common response codes.

Since most of the requested resources are read-only, the HTTP GET-method is used in most cases. Other HTTP methods POST and DELETE are used in all modifying cases, where POST is used for creating, updating and DELETE for removing the resource objects.

2.4.1.2 REST over HTTP

The SHIPLYS system is conceived as a distributed system, in which different components may be spread across different physical or virtual systems connected by network.

The mechanism to facilitate this shall be a REST-style API using HTTP connections [Fielding 2000] [1].

The REST API needs to consider best practices concerning:

1. Nouns – not Verbs convention.
2. Level of detail (keep the right balance between functional completeness vs. performance vs. API complexity).

3. API versioning.
4. Navigation support (e.g. HATEOAS).

2.4.1.3 HATEOAS

All actions performed within the framework should follow the HATEOAS (Hypermedia As The Engine Of Application State) principle [Fielding 2008] [2]. Simplified, this means that the responses from the server must contain hypermedia (hyperlinks) that guide the client to the next possible state(s) of the server application. Following this principle allows any client to fully interact (exploit the full functionality of the web-service) with the server without prior knowledge about the API-structure. The only prerequisite is the initial request-URL (root-URL of the web service).

Example:

The client sends a GET-request to the root URL of the web service:

```
GET /dbs HTTPS/1.1
Host: api.shiplys.com
Accept: json
```

A possible response from the server could be structured like this:

```
HTTPS/1.1 200 OK
Content-Type: application/json
{
	"id": "dbs",
	"href": "https://api.shiplys.com/dbs",
	"offset": 0,
	"limit": 10,
	total: 2,
	"items": [
		{
			"id": "database1",
			"href": "https://api.shiplys.com/dbs/db1"
		},
		{
			"id": "database2",
			"href": "https://api.shiplys.com/dbs/db2"
		}
	]
}
```

By identifying the content of the *href-properties* as hyperlinks, the client can identify the next possible steps (application states).

2.4.1.4 Resources

The generic resource class provides the following overall properties inherited by other specific resources:

id: Global entity identifier.

user: User processing the resource.

href: URL of the resource itself to ensure the HATEOAS principle.

2.4.1.5 Collection Resources and Pagination support

In order to limit web traffic and increase user readability, the API shall be able to support pagination of Collection Resources. This means, whenever a requested resource contains a collection (that potentially has a huge number of elements), the API shall return by default only a selected subset of this resource, if not stated otherwise by request-parameters. To guarantee unambiguousness, every returned collection shall contain the parameters:

limit: Number of elements returned (the size of the returned collection “window”).

offset: Index of starting point of the selection (position of the “window”).

total: Number of elements in total (all elements in the collection).

Furthermore, the Collection Resources provide a time stamp property in order to ensure an unambiguous working state.

2.4.1.6 Protected Resource Access via OAuth 2.0

Some of the data stored in the services or transported via the API are confidential and need to be restricted in access as well as protected while transported. OAuth 2.0 [Hardt 2012] [3] shall be used to guarantee data (access) security. OAuth 2.0 is an industry standard that addresses these problems via role separation (resource owner, client, authorization server, resource server) and different grant types. Focus is on simplicity for application developers and end users.

2.4.2 Meta Data Service (read-only mode)

Provides detailed information about the data structures (also referred to as the data model) used for representing ship design data, life-cycle data as well as design process states. It should be noted that the functions of this service are not necessarily specific to the ship design domain, but rather for representing data structure information in general.

2.4.2.1 Generic Data Schema handling

The Meta Data Service should be capable of handling multiple schemas and versions thereof.

The schema is specified by its name, description, revision, state and copyrights/intellectual property information. The state will express whether a schema is experimental, under test, in production use, deprecated or withdrawn.

The schema considers the following 4 categories of data structure information:

- Basic Data types: represent commonly used data types to define the properties of the entity types. These should include:
    - Boolean.
    - Integer/Long/Count.
    - Real/Double
    - Measure (further detailed by quantity and default unit)
    - Date, Time, Datetime
    - Enumerated values
        - Example (from ISO 10303-218 2004, in EXPRESS syntax):
        - `TYPE plate_type = ENUMERATION OF (bent_plate, bracket_plate, build_template, chamfer_plate, clip_plate, flat_bar, knuckled_plate, shell_plate, standard_plate, swedged_plate);`
    - String
- Sequences: Lists, Sets, Maps.
- Entity/Structured types: represent object types described by their properties. Entity types support multiple inheritances such that properties can be inherited from one or more other parent entity types. Entity types form the main content of a schema. An example (in EXPRESS syntax) is:
    - ENTITY plate inherits from steel part.
- Properties: represent the properties of an entity type, they are defined as basic data type or an entity type.

2.4.2.2 Process Model

The Meta Data Service delivers the read-only process model definition that underlies the SHIPLYS – Framework. It provides a formal description of the early ship design process, linking activities via transitions. Therefore, data entities are used to represent the particular activities, transitions, data exchanged (flows) between the activities and the executing mechanisms. As well as the general properties like the name and description the entity type activity contains the following properties:

- Transitions
- Inputs
- Outputs
- Controls
- Flows (Accumulation of Inputs, Outputs, Controls)

- Mechanisms
- Incoming activities (i.e. the set of activities linked by preceding transitions)
- Outgoing activities (i.e. the set of activities linked by succeeding transactions)

Each of the properties is also represented by a particular entity type with specific properties.

The retrieval of instances of the process model items enables clients like the Design Process Monitor to analyse the current status and the input needed to carry out next steps regarding the ship design, life-cycle or other tasks.

2.4.2.3 Ship Design Data Model

The Meta Data Service shall store the data model that is an essential underpinning of the SHIPLYS framework to represent both the design process as well as any design data of a ship. The Meta Data Service shall be able to store more than one (version of the) data model. If requested, the Meta Data Service shall return the formal definition of (single) data types to the client. The data model (data schema) shall contain resources such as:

- Schema: providing a detailed description of the selected data model (or schema version)
- Types: different data types of the data model such as (below only some examples are used)
    - Plates.
    - Compartments.
    - Spacing tables.
    - …
- Properties: data type to represent the properties of a type, such as:
    - Thickness
    - Density
    - Length
    - ...
- Enumerated type definitions.

2.4.3 Data Service (read-write/read-only modes)

The Data Service stores the actual data entity instances relevant to projects. A project contains all data that pertains to a particular design project.

2.4.3.1 Storage

The minimal functional level of the Data Service is to act as a directory or broker for data instances. In this mode, only a reference (e.g. URI/IRI) is kept as a reference to the actual data. In this mode, when an instance is created, the target URI is stored. On retrieval, the URL will be returned.

Optionally, the Data Service may provide storage capabilities for the instance data. For this purpose, a specific data container format (e.g. JSON, XML etc.) shall be used. In this mode, when an instance is created, its actual content is uploaded using such a data container. On retrieval, that data will be delivered in the same format as well. Note that this does not imply any kind of conversion to be provided.

2.4.3.2 Version Management

Data instances have a life-cycle. A specific instance is identified by a version (including a time stamp and origin). The combination of time stamp and origin is used as a unique version-based identification which ensures, that no duplicate version + origin combination can be created at any time. Instances may have multiple versions. Default retrieval shall always return the latest version.

Instances may have a time-to-live (TTL) indication. This allows older versions to be purged. Purging is an optional operation, which may be provided by implementing services. Objects that have passed their time-to-live indication will be subject to a garbage collection which may result in object deletion, once no live references exist. Client must accept that objects, whose TTL has expired may disappear.

Instance versions may be marked as obsolete (“retired”) or erroneous.

NOTE: there is no support for explicit deletion by clients to avoid various transaction and consistency issues and to support history tracking

2.4.3.3 Data State

Any data instance may have an associated data state.

This shall allow applying standard stage tags to any instance. For example, a calculation result may be marked as estimated, approved etc.

2.4.3.4 Data protection

Several levels of information/data criticality must be supported.

2.4.3.5 Organisation/Context

The Data Service stores and delivers the actual data instances related to an organisation. Additionally, common data may be required for general use, e.g. providing the context for a specific organisation. If requested, it returns the data entity to a client. A client must be able to add or update such data instance to the Data Service, if it is authorized.

2.4.3.6 Ship

The Data Service must be able to store and deliver all data instances that are required to specify the ship and its parts. If requested, the Data Service returns the requested data entities to the requesting client. A client must be able to post such data instances to the Data Service, if it is authorized. The data instance of type “ship” shall have a link to the underlying data schema stored in Meta Service.

2.4.3.7 Design Process

The Data Service must be able to store all information relevant to describe the state of the design process corresponding to the Process Model. Authorized clients must be able to retrieve information and change the Design Process State via communication with the Data Service.

2.4.3.8 Operations

This sub-section covers the main operations that shall be provided by the Data Service.

Create Instance

The Data Service shall support the creation of a new instance of an entity type using the POST operation. Depending on the supported storage mode, either an URI or a data container (and optionally an URI) must be provided by the client.

Search for Instance

The Data Service shall allow the search for instances by search parameters such as name, id, property, extent, etc. Search parameters can be combined in any order and number. The Service should also allow search for Instance based on version and origin, using wildcards if necessary.

List Extent

The Data Service shall be able to list the extents of an object if queried.

Assign Data State

The Data Service shall be able to assign/update the data state to a selected data object instance.

Inquire Data State

The Data Service shall be able to inquire the data state of a selected data object.

2.4.3.9 (OPTIONAL: Referencing Data Type)

All entities stored by the Data Service shall contain an optional property referencing the data entity type in the Ship Design Data Model (Data Schema). The decision whether the retrieved data entity has this property shall be made using a query parameter.

2.4.4 Software Registry Service (read-write/read-only modes)

2.4.4.1 Software component

The software registry provides a central place to accept requests for registering or searching already registered software components. In detail, the software registry must provide the general functions mentioned below.

Software characteristics

The Software Registry must provide the functionality to characterize software. Characteristics are defined as:

- **Functional Categories:**
  - Functional Scope (Keywords).
  - Activities covered.
  - Input Data.
  - Output Data.
  - (...)
- **Technical Properties:**
  - Runtime requirements.
  - Licensing scheme.
  - Maturity (e.g. new development, prototype, partner's background commercial software, third party commercial and third party open source).
  - An indication of time.
  - (...)
- Performance classification.

The Software Registry should provide a mechanism to estimate the expected time needed to process the specified task. Ideas for that kind of mechanism are:

- Range (longest time ever measured and shortest time ever measured).
- Correlation with one driving parameter (e.g. elements in FE-calculation).

2.4.4.2 Register Software components

The software registry must be able to process requests for registering new software components. This request must at least contain:

- The name of the software component.
- A description.
- The functional scope (keywords).
- Supported tasks (activities and the data flows within the ISO 10303-based SHIPLYS Activity Model).
- Its runtime instances, whereas a specific ID is generated internally.

The runtime instances contain the information regarding the status, which can be activated or deactivated, a link to the related job service as well as information regarding the license type and platform.

In addition, further properties regarding the meta information of the new software to be registered can be determined.

These properties comprise information about

- Version.
- Originator.
- Distribution.
- Functional description.
- Information references.
- License type.
- Applicable fees.
- Platforms.
- Interfaces.
- Programming language and documentation.
- (...)

The software Registry must respond to the request by registering the software component.

After the registration, the software properties shall be subject to search requests and shall be returned by the software registry when it fits to the search parameters. Authorisation must be appropriate to carry out any such searches.

A software registration may also be updated, e.g. by registering additional runtime instances, adding additional/remove obsolete supported tasks etc. Furthermore, the status of a runtime instance can be changed or a runtime instance can be completely removed, whereas in both cases the request is rejected, if the runtime instance is currently processing a job.

2.4.4.3 Search for software components

The software registry must support searches for registered components using one or more different search criteria. For example:

- It shall return a list with all registered tools that allow calculation of some data collection/data flow

- Return information about specific software
- Complex queries (data certainty, underlying method…)
- Some tolerance in spelling of software names and functional criteria.

2.4.4.4 Relation to the job service

The relation between the software registry and the job service is provided by an URL property within each of the specific software runtime instances. This URL points to the job service responsible for the execution of the selected software runtime instance, whereas a job service may manage several software runtime instances.

2.4.4.5 Activate/Deactivate registered software

The Software Registry must provide the functionality to activate or deactivate registered software since it might be decided that this software should not be used temporarily or indefinitely. The software information nevertheless must be maintained in order to be able to identify software components that have been used previously to produce data. A software can be assumed to be deactivated if it has no runtime instances at all or all its runtime instances have been deactivated.

2.4.5 Job Service

The Job service is provided to support interaction with SHIPLYS end-points supporting the execution and monitoring of registered software. As part of the software registration, access details to the responsible Job service must be provided. Details about the used version of data shall be stored.

It must be possible to start a software component via a common mechanism that hides the details of the actual integration mechanism applied as well as the technical details on how to run the software. It shall also check the availability of a software component in terms of license status and system availability or load.

As part of the start request, references to the related input data shall be accepted and consumed.

Note that running applications will always be asynchronous. Once a job is started, the client shall be provided with some sort of “job handle” which enables the calling client side to communicate directly with the executing end-point running the software.

2.4.5.1 Job Service End point

Each real or virtual system providing a service to access and execute registered software represents a Job Service End Point, which must be accessible via a unique URI.

Any such end point may support one or more different types of registered software with a corresponding executing profile (capacity in terms of available concurrent instances and support hardware capacity).

2.4.5.2 Software Runtime Instance

Any executing software job will be realised using a runtime instance of a specific software component on the target end point. Thus, every job executed by a specific job service will be associated with a specific software instance.

2.4.5.3 Job

Each job will have a unique set of characteristics including a globally unique ID, software instance, runtime settings, input parameters, expected output parameters, whether the job is interactive, embedded, if it requires queueing etc.

2.4.5.4 Job Creation

Creates a new job of a specified software component. Provides a “handle” (i.e. a URL) suitable for future reference to the job. Furthermore, information regarding the status, start and end date is considered.

2.4.5.5 Job Cancellation

Informs the Job service about the request to cancel an executing job.

2.4.5.6 Job Status

Retrieves information about a job status, whether running, paused, completed, aborted, restarted, cancelled or obsolete. If results are available, the appropriate references to the stored data shall be available. We will consider maintaining a log of completed jobs.

Jobs having the status obsolete could be deleted automatically after their results have been stored.

2.5 Use Cases

NOTE 1: in the URL/URI/IRI examples ~ replaces the URL prefix such as https://<some-host>/v1/

NOTE 2: the API version is reflected by a version path element such as v1, v2, …

NOTE 3: some sample URL may contain special characters that need to be mapped in valid URLs. For readability, this has not been applied.

NOTE 4: request and response content is not shown in a technically and syntactically precise notation in this section, but only for the purpose of illustration.

2.5.1 Meta Data Service Use cases

2.5.1.1 Retrieve general schema information

The client retrieves the general information regarding the used schema with respect to the current project.

Example of the request (assuming that a schema called SHIPLYS-SCHEMA exists):

~/meta/dataSchema/SHIPLYS-SCHEMA

```
Response:
Name: SHIPLYS-SCHEMA
Description: Schema to describe life-cycle relevant data during the early ship design
Revision: 1234
Copyright: Copyright © 2019, The SHIPLYS Consortium
```

2.5.1.2 *List entity types*

The client retrieves all entity types provided by the SHIPLYS framework sorted alphabetically. In this way, it is possible to get an overview which entity types do already exist and can be used. Furthermore, schema versioning can be considered.

Example of the request:

~/meta/dataSchema/{schemaID}/v1/types/

```
Response:
assembly_node, block, compartment, design_loading_condition, pipe, plate,
process_template, zone, …
```

2.5.1.3 *Lookup entity type by name / Retrieve entity details (e.g. inheritance)*

The client retrieves a particular entity type in order to obtain detailed information.

Example of the request:

~/meta/dataSchema/SHIPLYS-SCHEMA/types/compartment

```
Response:
id: SHIPLYS-AAM-compartment
href: “...”
name:
version:
properties:[
      {
            "name": "parameter1",
            "href": "..." ,
            "objectInstance": "..."
      },
      {
            "id": "parameter2",
            "href": "..." ,
            "objectInstance": "..."
      }
]
timestamp: 2017-05-17T14:30+02:00[Europe/Paris]
```

```
Copyright: Copyright © 2019, The SHIPLYS Consortium
```

2.5.1.4 *Retrieve entity type properties*

The client retrieves the supported properties of a particular entity type.

Example of the request:

~/schema/SHIPLYS-DESIGN/types/compartment/properties

```
Response:
boundary, cargo_positions, center_of_volume, coating, noise_classification, tightness...
```

2.5.1.5 *Lookup property by name / Retrieve property details*

The client retrieves the details of a particular property.

Example of the request:

~/schema/SHIPLYS-DESIGN/types/compartment/properties/tightness

```
Response:
Id: tightness
Description: ...
Type: {
      id: tightness_type
      href: …
      values: [air_tight, fume_tight, gas_tight, non_tight, oil_tight, water_tight,
      weather_tight
      ]
}
```

2.5.1.6 *Retrieve general process model information*

The client retrieves the general information about the process model with respect to the current project.

Example of the request:

~/meta/processModels/SHIPLYS-AAM

```
Response:
Name: SHIPLYS-AAAM
Description: The Activity Model used in SHIPLYS
Revision: 1234
Copyright: Copyright © 2019, The SHIPLYS Consortium
```

2.5.1.7 *Inquire the list of all parameters in the process model*

The client sends a request to retrieve a list of all parameters in the selected process model.

Example of the request:

~/meta/processModels/SHIPLYS-AAM/parameters

```
Response:
id: SHIPLYS-AAM-parameters
limit: 10
offset: 0
total: 25
href: “...”
items:[
	{
		"name": "parameter1",
		"href": "..." ,
		"objectInstance": "..."
	},
	{
		"id": "parameter2",
		"href": "..." ,
		"objectInstance": "..."
	}
]
timestamp: 2017-05-17T14:30+02:00[Europe/Paris]
Copyright: Copyright © 2019, The SHIPLYS Consortium
```

2.5.1.8 *Inquire the list of all outputs of one specific activity*

The client sends a request to retrieve a list of all outputs of one activity.

Example of the request:

~/meta/processModels/SHIPLYS-AAM/activities/A122/outputs

```
Response:
id: A122-outputs
limit: 10
offset: 0
```

```
total: 25
href: “..”
items:[
	{
		"name": "output1",
		"href": "..." ,
		"objectInstance": "..."
	},
	{
		"id": "output2",
		"href": "..." ,
		"objectInstance": "..."
	}
]
timestamp: 2017-05-17T14:30+02:00[Europe/Paris]
Copyright: Copyright © 2019, The SHIPLYS Consortium
```

2.5.2 Data Service Use Cases

2.5.2.1 Lookup ship/project by name

A user wants to retrieve information about a specific project/ship. He/she sends a request and the Data Service answers with the data element of type e.g. “ship” (or “project”).

Example of the request:

~/data/exampleOrganisation/name=ship_xyz

```
Response:
<URL to ship>
```

2.5.2.2 Retrieve extent of a type

A user wants to get the extent (the complete set of instances of a type) of a specific entity (such as ship, compartment, loading condition, etc.) and sends a corresponding request. The Data Service responds with the set of instances of that type that contained the project. E.g., if entity is of type ship, then the result would be a set containing only a single instance: the ship. If the entity type is compartment, then the result would be the set of compartments defined for the ship.

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/plate

Response:

A set of <URL>s to all plates

2.5.2.3 Lookup / Find / Query single entity or entities

A software client system wants to retrieve the compartment(s) that exists between given x,y,z coordinates. It sends a query of type “return all compartments with extents between...”

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/compartment?lb=(10.0m,20m,0)&ub=(16.5m,0m,2525mm)&mode=intersect

Response:

A set of <URL>s to all compartments intersecting with the given volume

A user wants to retrieve all plates that extend from .. to .. and have an actual thickness below a limit value. A corresponding request to the Data Service delivers a set of plates that match the criteria. The value returned will be in relevant standardised units for the material in question.

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/plate?ub…?lb…&thickness=<7mm

Response:

A set of <URL>s to all plates

2.5.2.4 Retrieve instance data

The user is in possession of an entity and wants to get its actual data. He/she sends a request of type “Get data of instance” and the Project Data Service delivers the data of the specified entity.

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/plate/A100-23028-1P/v/2016-10-20T151000/

Response:

A JSON data set with all data details of the instance

2.5.2.5 Store new entity instance or new version of an instance

A retrofit project requires the storage of a new scrubber. A request is sent and processed by the Data Service creating a new scrubber object.

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/product_component/MainEngine–SC-4/?....<data details>

```
Response:
The <URL> to the newly created instance
```

NOTE: POST method allows a complete JSON data package to be uploaded.

For a new version of an instance, the request is shown below.

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/product_component/MainEngine–SC-4/v/?....<data details>

```
Response:
The <URL> to the newly created instance version
```

Note: POST method allows a complete JSON data package to be uploaded.

2.5.2.6 Assign/Update entity instance state

Within an iteration loop, the dimensions of a profile have been confirmed. The user/client/software sends a request to change the status from estimated to calculated or approved. The Data Service updates the data state.

Example of the request:

~/data/exampleOrganisation/ships/ship_xyz/xtents/profile/A100-23028-S12/v/2016-10-20T151000/?state=calculated

```
Response:
The updated state information for the target profile
```

2.5.3 Software Registry Service Use cases

2.5.3.1 Register new software

A user discovers a new software component which he/she wants to register. As a consequence, a registering request is sent to the software registry. The request must contain at least the name, the functional profile of the new component and the appropriate runtime instances containing a reference to a specific job service they are implemented by. After the registration, the software must be available within the framework context. This function will be subject to various authorisation checks. Note that most likely not every user will be allowed to perform this function.

It may also be considered to make this a two-stage process, requiring some kind of approval e.g. by a project or system manager.

Example of a POST request:

~/swreg/software

```
Part of the request Content:
        name: Ultimate Ship Designer
        supportedTasks: A122-Create preliminary design, A1221-Create preliminary hull form
        runtimeInstances:
                runtimeInstance:
                        jobService: ~/jobs
                        statusActive: true
                        licenseType: Standalone
                        applicableFees: 2000 €
                        supportedPlatform: Windows 10
        version: 1.6
        functionalScope: []
        etc.

Part of the response Content:
        Software successfully added
        id: softwareId
        href: ~/swreg/software/softwareId
```

2.5.3.2 Search for software based on functional characteristics

Example: A preliminary hull form has been designed. A probabilistic damage stability calculation should be executed. Only software should be selected that is expected to be able to return the result within a specified period of time. The user sends a request with corresponding criteria and the Software Registry answers with a (possibly empty) list of tools that fulfil the specified requirements.

Apart from tasks to be supported, the following request contains a limit of 10 for number of returned results. A default offset of 0 is implied within this request. Furthermore, the query parameter condition “statusActive” is used in order the get results containing only activated software which can be used immediately. It is also possible to get more information at once without using the returned link but by the use of a further query parameter “expand”.

Example of a GET request:

~/swreg/software?limit=10&tasks=damage stability&calculation time<5h&statusActive=true

```
Part of the response Content:
        id: softwareId
        href: ~/swreg/software/softwareId
```

2.5.3.3 Search for software based on name

A user looks for a specific tool with a known name and wants to acquire information about this tool. The Software Registry responds to the search request by returning the selected tool reference and information.

As mentioned above, in order to receive more information at once it is possible to use the expand query parameter as shown in the following example.

Example of a GET request:

~/swreg/software?expand&limit=10&name=Ultimate Ship Designer&statusActive=true

```
Part of the response Content:
    id: softwareId
    href: ~/swreg/software/softwareId
    name: Ultimate Ship Designer
    supportedTasks: A122-Create preliminary design, A1221-Create preliminary hull form
    runtimeInstances:
        id: instanceId
        href: ~/swreg/software/softwareId/instances/instanceId
    version: 1.6
    functionalScope: [...]
    platforms: Windows 10, Linux
```

2.5.3.4 Software usage negotiation

The software registry has to be able to perform the software usage negotiation with the user. This means it has to provide the user with a fitting software list and react to the user's selection by activating the selected software.

2.5.3.5 Activate & Run Software component

After the selection of the software, the software registry shall provide the appropriate means to run the selected software. This requires the necessary authorisation for accessing the input data, executing the software as well as archiving of the results.

The following example shows how to change the software status which depends on the status of its runtime instances.

Example of a POST request:

~/swreg/software/{softwareId}/instances/{instanceId}

```
Request Content:
    jobService: ~/jobs
    statusActive: true
    licenseType: Standalone
    applicableFees: 2000 €
    platform: Windows 10
```

```
Response Content:
        Runtime instance successfully updated
```

2.5.4 Job Service Use cases

2.5.4.1 Create a job

Each of the jobs is related to a specific software runtime instance, whereas several jobs may be assigned to one runtime instance. For that reason, in order to create a new job, a software runtime instance has to be provided.

Example of a POST request:

~/jobsvc/instances/{instanceID}/jobs

```
Part of the request Content:
        start: 2017-01-01 15:45:00
        end: not available yet

Part of the response Content:
        Job successfully created
        status: created
        id: jobId
        href: ~/jobsvc/instances/{instanceID}/jobs/{jobId}
```

2.5.4.2 Retrieve jobs by a software runtime instance

By requesting a software runtime instance a link to the related jobs is provided shown within the following example, which connects the software registry service with the job service.

Example of a GET request:

~/jobsvc/instances/{instanceID}/jobs?expand&limit=10

alternative request via query parameter:

~/jobsvc/jobs/?instance={instanceID}&expand&limit=10

```
Part of the response Content:
        id: jobId
        href: ~/jobsvc/instances/{instanceID}/jobs/{jobId}
        status: running
        softwareRuntimeInstanceId: 1
        start: 2017-01-01 15:45:00
        end: not available yet
```

2.5.4.3 Status change of a job

The status of a job can be running, paused, completed, aborted, restarted, cancelled and obsolete. The following example shows an update of a job after its completion.

Example of a POST request:

~/jobsvc/jobs/{jobId}

```
Request Content:
        status: completed
        softwareRuntimeInstanceId: 1
        start: 2017-01-01 15:45:00
        end: 2017-01-01 17:00:00

Response Content:
        Job successfully updated
```

2.6 Simple Integration Example

2.6.1 Introduction

This section illustrates an example of the integration of two software tools: the RSET early stage design tool and a simple LCC spreadsheet tool. The integration method used in this example might be adapted to the other design and life-cycle analysis tools, so that it might serve as a basic prototype for integration.

2.6.2 Approaches to Integration

As indicated in section 2.1.1.4, integration may be approached in several ways. The case of two software tools is considered here, but the techniques used could potentially be extended to allow a greater number of tools to exchange data and operate in conjunction with each other.

A given pair of software tools may either interface with each other directly (where they have been designed to do this, or where plug-in code can facilitate this functionality), or indirectly via some other means (e.g. glue code). The best option out of these alternatives will vary, depending on the type and functionality of the tools, the amount of data to be exchanged, etc. For instance, where two tools must be repeatedly executed many times in quick succession, a direct interface may be the best approach, after which the final results are returned to the SHIPLYS platform. In other cases, it may be favourable for the platform to act as an intermediary between the tools (for example, to ensure data consistency or to perform some error-checking function during exchange of data).

2.6.3 Selection of an approach

There are a number of key considerations when determining the best approach for integrating any two software tools. Firstly, the prime objective must be defined;

- Are the two software tools performing some kind of optimisation?

- Is some iterative procedure required to produce an outcome compatible with user specified constraints (e.g. keeping a component of cost within a specified budget)?
- Are the tools designed in such a way that a particular method for integration is simpler to implement?

Answers to these questions will help determine whether the REST interface or some other simple method for software integration is the preferred approach.

The example case described here illustrates integration of two tools via the SHIPLYS platform.

2.6.4 General Workflow

The following example workflow shows the steps involved in transferring data from one tool to the other via the SHIPLYS platform.

1. User activates tool A via the platform
    a. Required data is loaded from the platform into tool A
2. User completes required tasks in tool A
3. User saves/exports results of tool A
4. Platform captures results
5. Tool B is activated (via user action or automatically by platform)
    a. Results from tool A (and other required data) are loaded via platform into tool B
6. User completes required tasks in tool B

2.6.5 Benefits of this integration method

The platform handles all data and can therefore:

- Check for completeness
- Check for consistency with existing ship model data generated from previous tools
- Merge output data from one tool with data from other sources to be fed to the next tool in the workflow

2.6.6 Brief description of RSET

The Rapid Ship Evolution Tool (RSET) is an early stage design tool for determining compartment arrangement within a given hull form and superstructure space, subject to user-specified constraints and preferences for compartment placement. It will serve here as a simple example of an early stage design tool to be integrated with a cost estimation tool.

The tool requires as input a vessel hull form (in Wavefront .obj file format), but may also be supplied with some set of the following inputs:

- A list of compartments to be arranged (.csv).
- A list of bulkhead positions (.csv).
- A list of deck positions (.csv).
- A definition of the superstructure decks (.csv).
- A list of compartment arrangement constraints (.csv).

2.6.7 Brief description of LCC tool

The LCC tool consists of a set of statistical relationships based on historical cost data which relate the input parameters (vessel dimensions, total volume of compartments within each WBS element, etc.) to various cost components of the LCC.

The LCC tool will take as input the following:

- The output vessel design produced by the RSET tool, including:
  - Vessel dimensions.
  - Hull form.
  - Compartment details (type, size).

These parameters will serve as input data to predict construction costs based on a parametric cost estimate. If appropriate cost data is available, maintenance, operation and decommissioning costs might also be included in order to produce a total life cycle cost estimate. Otherwise, the tool may simply estimate construction cost, which can be fed back to the SHIPLYS platform and combined with estimates from other tools to create a life cycle cost estimate.

2.6.8 Integration example

Following the general workflow given in section 2.6.4, integration of RSET and the LCC tool may be achieved in the following manner:

1. User activates tool A; required data loaded from platform into tool A

In general, the means by which the input data is provided to the tool will vary:

- Launch command opens an appropriately formatted file with the tool.
- Launch command runs the tool and then activates a data import function within the tool.

This step may require some user interaction – for example, the user may need to specify what data should be provided to the tool.

In the current version of RSET, the user must interact with the tool through the GUI interface to specify the inputs not already supplied as input files and run the compartment arrangement solver.

2. User completes tasks in tool A

Some tools may not require user interaction to perform their function, but in most cases, some degree of user supervision (at minimum) or input will be required.

Specification of compartment arrangement constraints and preferences might be provided via input files in future revisions of RSET, but currently is performed by the user within the program interface.

3. User saves/exports results of tool A

If the tool requires user interaction, the user may need to export the results from the tool. If user interaction is not required, output/export of the results could be automated, and this step skipped.

Once an arrangement is generated in RSET, the user may export attributes of this arrangement to an output file (.csv containing vessel dimensions, positions of compartments, etc.).

4. Platform captures results

Once the output of tool A is finalised, the SHIPLYS platform should capture the results. Depending on the tool used, this might be achieved in one of the following ways:

- Data is transferred directly from the tool to the platform via data stream

- Data is imported into the platform from an exported file

In this example case, the SHIPLYS platform could simply read the data from the file exported in the previous step.

5. Tool B is activated - Results (and other required data) are loaded from tool A, via platform, into tool B

This step is essentially the same as step 1, but the tool may be activated automatically, depending on the functionality required. For example, a pair of tools may be alternately executed repeatedly to perform some goal-seeking or optimisation function. In that case, the user would specify a goal, constraints and termination conditions and then activate the process, which should then proceed without user intervention until the termination conditions have been reached.

If user interaction is required, the process continues in the manner described in steps 2, 3 and 4.

In this example case, the LCC tool has only simple data import/export functionality via reading/writing csv files, but as it is based on an Excel spreadsheet it may be readily modified (using custom VBA code) to interface with the SHIPLYS platform and/or RSET in a more direct manner.

Using the import/export functionality only, the SHIPLYS platform can activate the LCC tool and specify the location of a csv file for import. On launching the tool, the file is imported automatically based on functionality built into the tool, at which point the tool will proceed automatically to generate a cost estimate (based on the ship design parameters provided in the imported file), export this result to a csv file, then terminate. On receiving a signal from the tool indicating that it has terminated successfully, the SHIPLYS platform will then import the cost estimate from the csv file.

2.7 SHIPLYS Integration Suite mock-up

The figures below show a few sample screenshots of the mock-up SHIPLYS platform user interface. The screens show a few steps at different stages through the process of defining the early stage vessel design, through to some of the later stage steps including cost estimation and production planning.

It should be noted that at the time of writing, the particular software tools to be used for each activity in the workflow have not been finalised – those tools shown in the mock-up are examples only.

2.7.1 User interface overview

The overall workflow is divided into modules, with sub-functions within each module; these are predominantly based on the first and second level activities of the SHIPLYS Activity Model.

The modules are accessed by activating the buttons on the left hand side of the interface. Initially, only some of the modules are accessible to the user. Other modules become enabled as the design progresses and the required data and inputs for those modules are defined. Different functions within each module are contained in tabs (selected at the top of the screen).

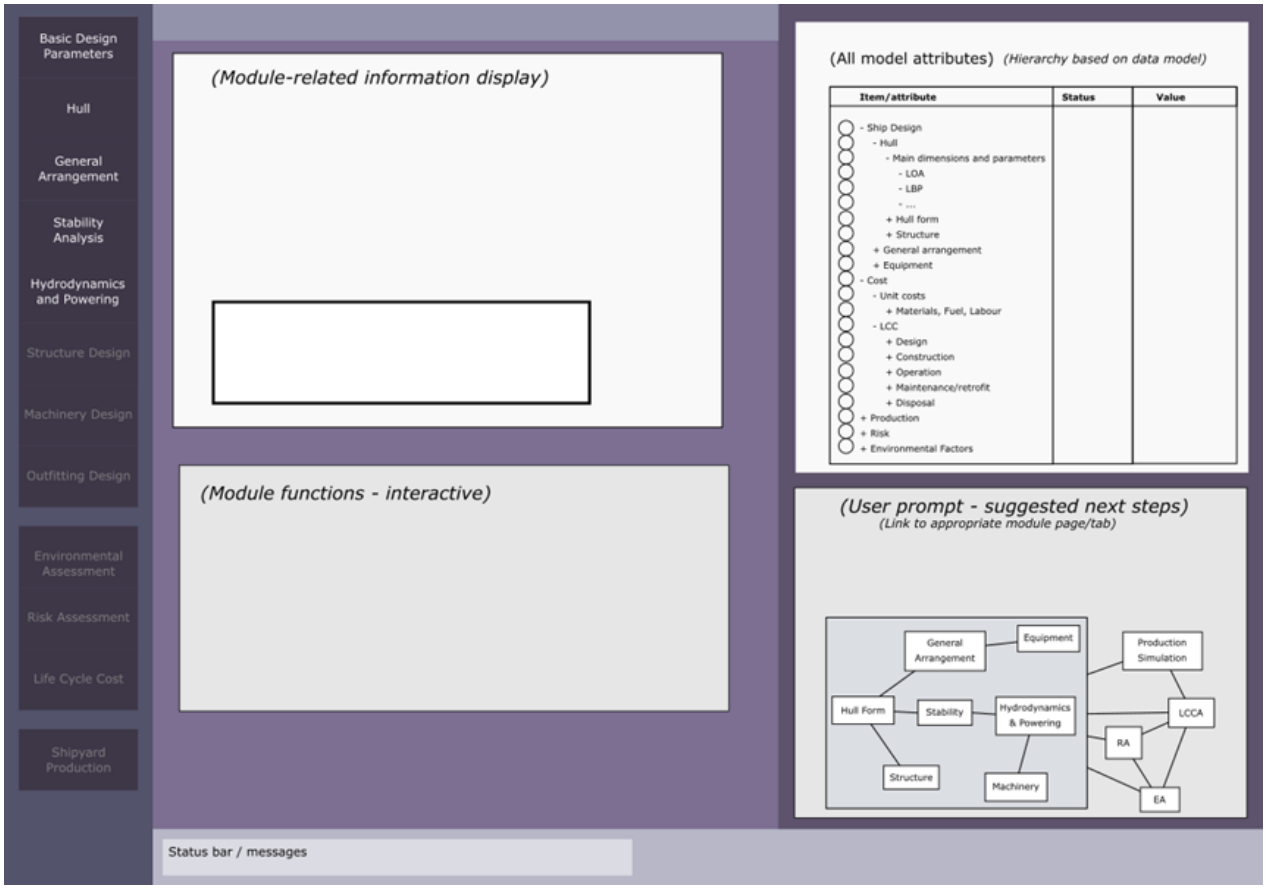

Figure 8: Mock SHIPLYS Integration Suite Interface

Information on the user's progress through the workflow and the completeness of the design is indicated in the right-hand pane: the top portion contains an expandable tree of all elements in the ship model and indicates their status and current values, while the lower portion prompts the user on which steps in the workflow they should ideally perform next and provides links to the relevant tabs. An interactive graph showing progress through the workflow and the status of the various steps is shown in the lower portion of the pane. Both sections employ red-amber-green visual indicators to indicate the status of an element.

The middle pane shows information (top) and user actions (bottom) relating to the currently selected tab (i.e. a sub-function within a module). The format of these will differ between different sub-functions and modules, depending on what information needs to be displayed and what actions the user needs to perform.

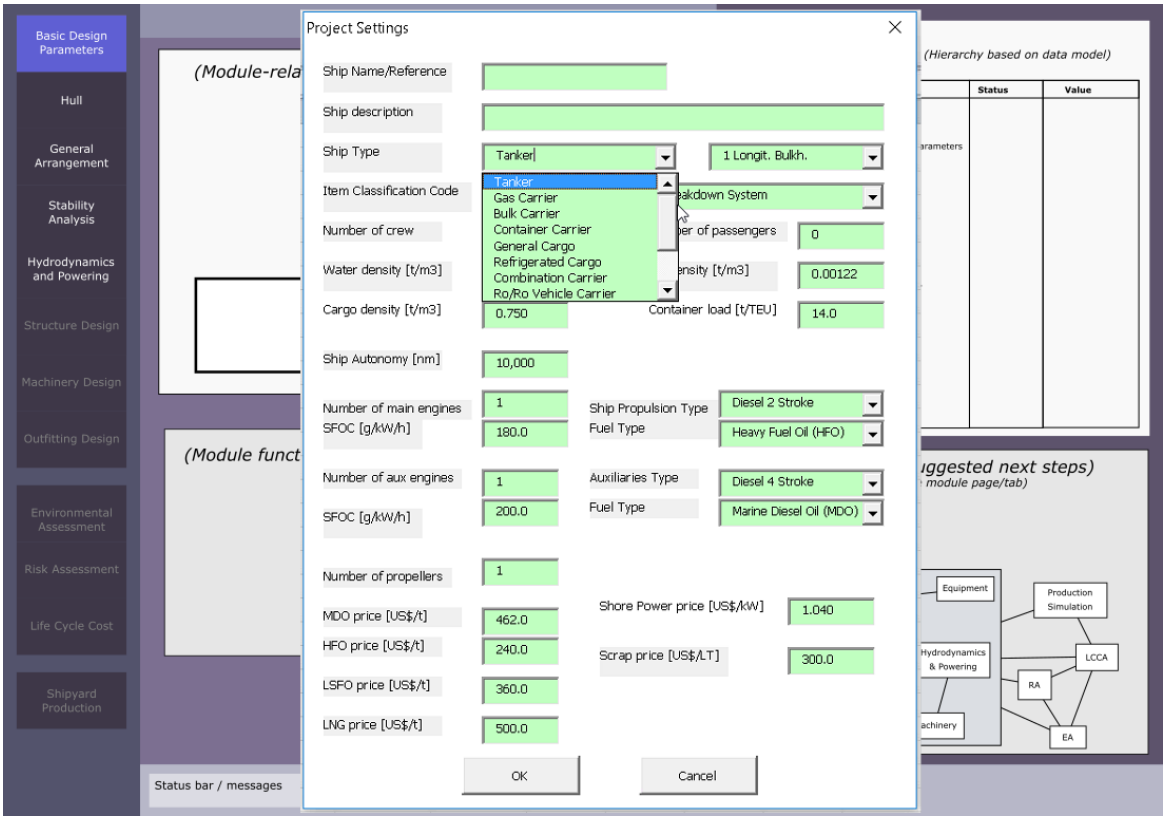

Figure 9: Start-up screen with prompt for initial settings (from IST spreadsheet tool)

On starting up a new project in the interface, the user is presented with a start-up screen as shown in Figure 9 above, prompting them to specify initial basic information about the vessel design. The settings screen is taken from a spreadsheet tool constructed by IST which may form part of the SHIPLYS tool chain.

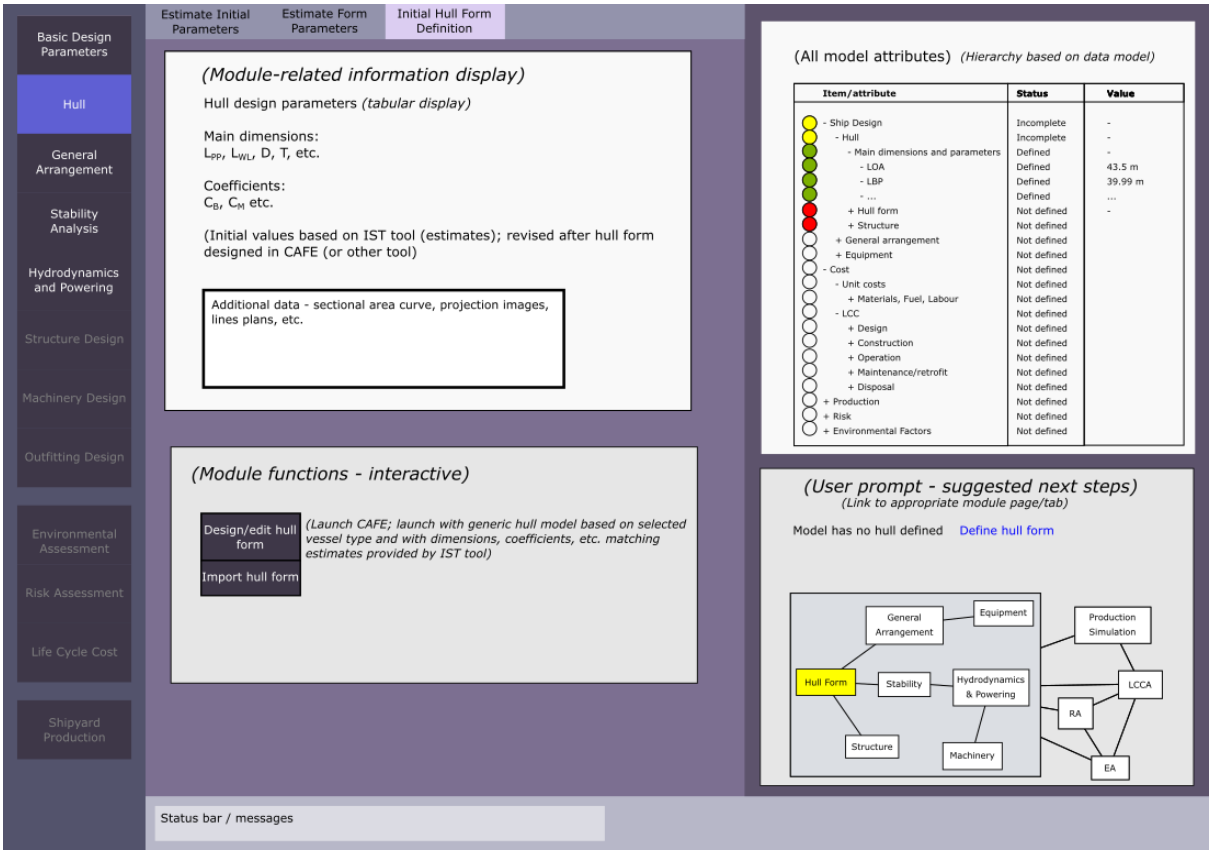

Figure 10: Hull form definition tab of the hull module

Prompts appear in the bottom-right pane of the interface, indicating to the user the next logical steps in the workflow and linking to the relevant modules. The user proceeds by selecting links in the prompt pane or by selecting a module and sub-function manually using the buttons and tabs. The user has ultimate control over the sequence of steps in the workflow – the prompts only serve as guidance and may be disregarded if the user prefers or requires a different approach.

After providing estimates of the initial ship and hull parameters (either manually or using the IST spreadsheet tool), the next step is to create the hull form (using CAFE [4]).

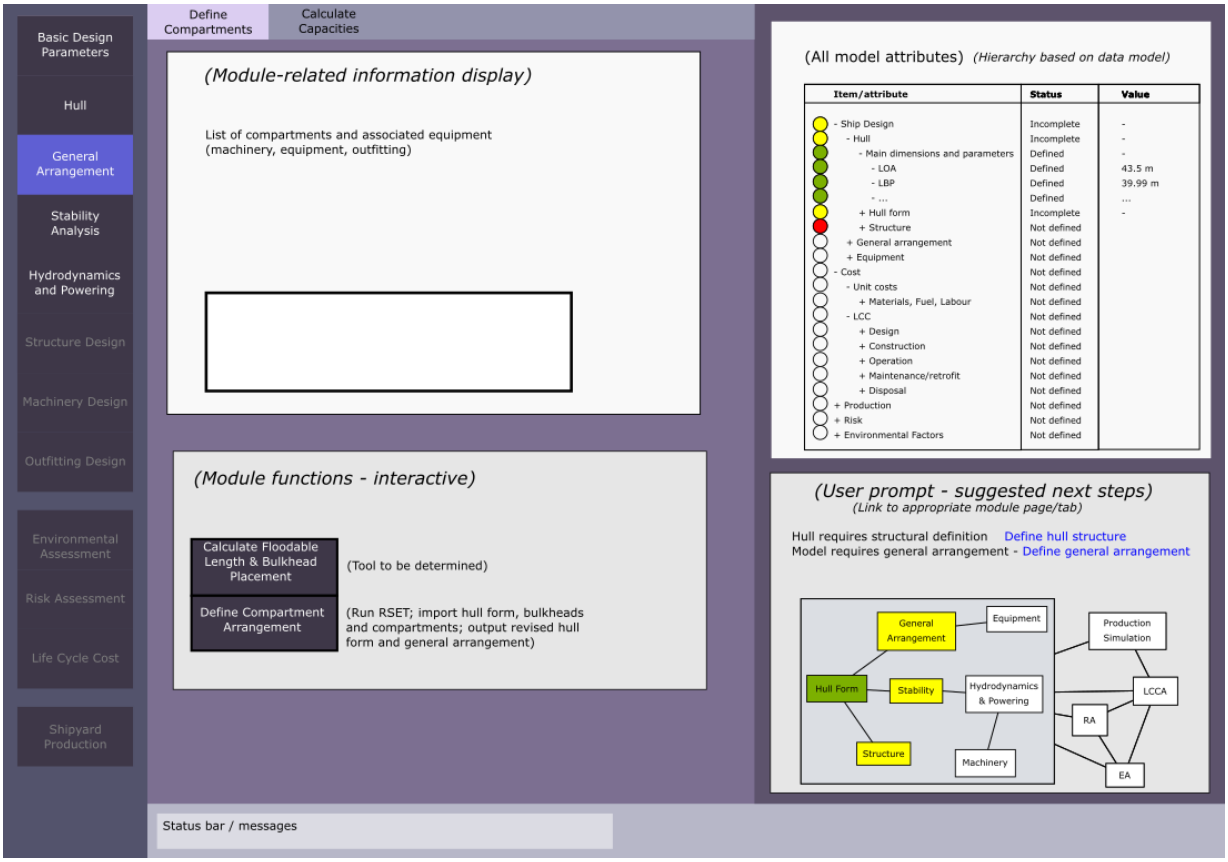

Figure 11: General arrangement module

Once the hull form has been created (or imported from another tool) and the required compartments have been determined, the general arrangement can be determined. At this stage, the user can also perform stability and structure related analysis in the appropriate modules.

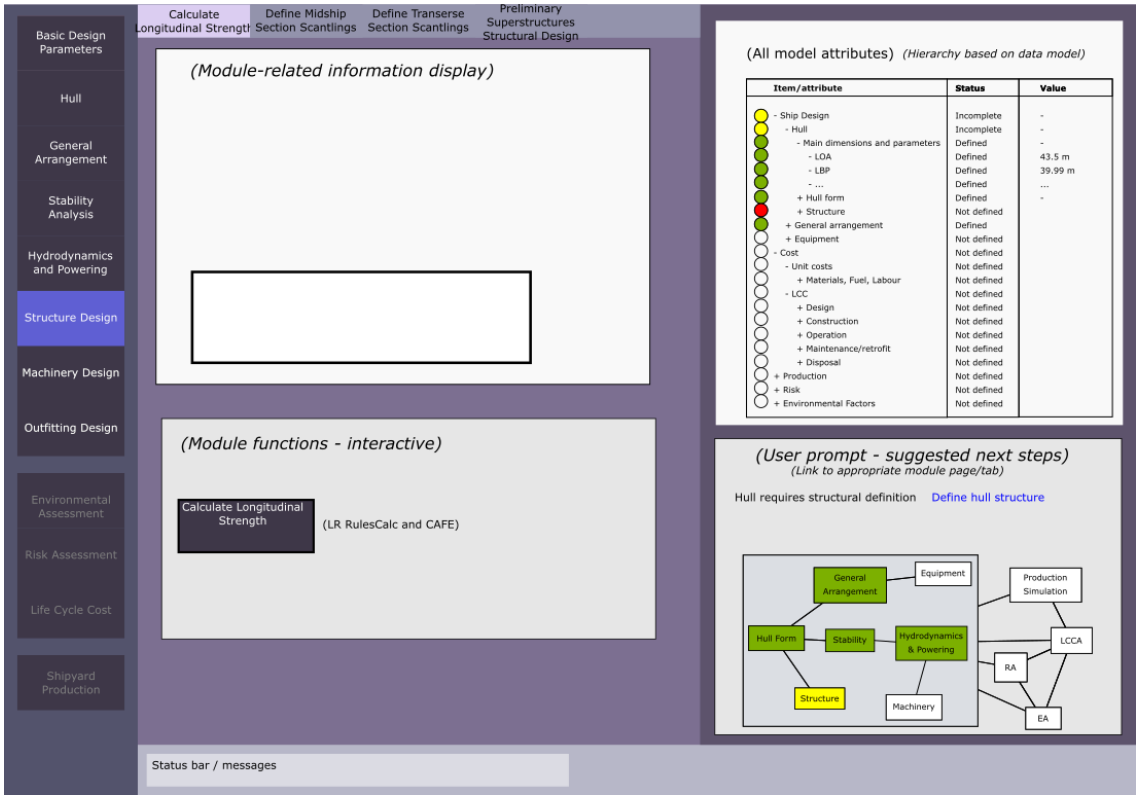

Figure 12: Structural analysis module

LR RulesCalc [5] might be used in conjunction with CAFE for most functions within the structural analysis module. As more of the model is defined, new modules become accessible to the user (i.e. the module buttons on the left become enabled).

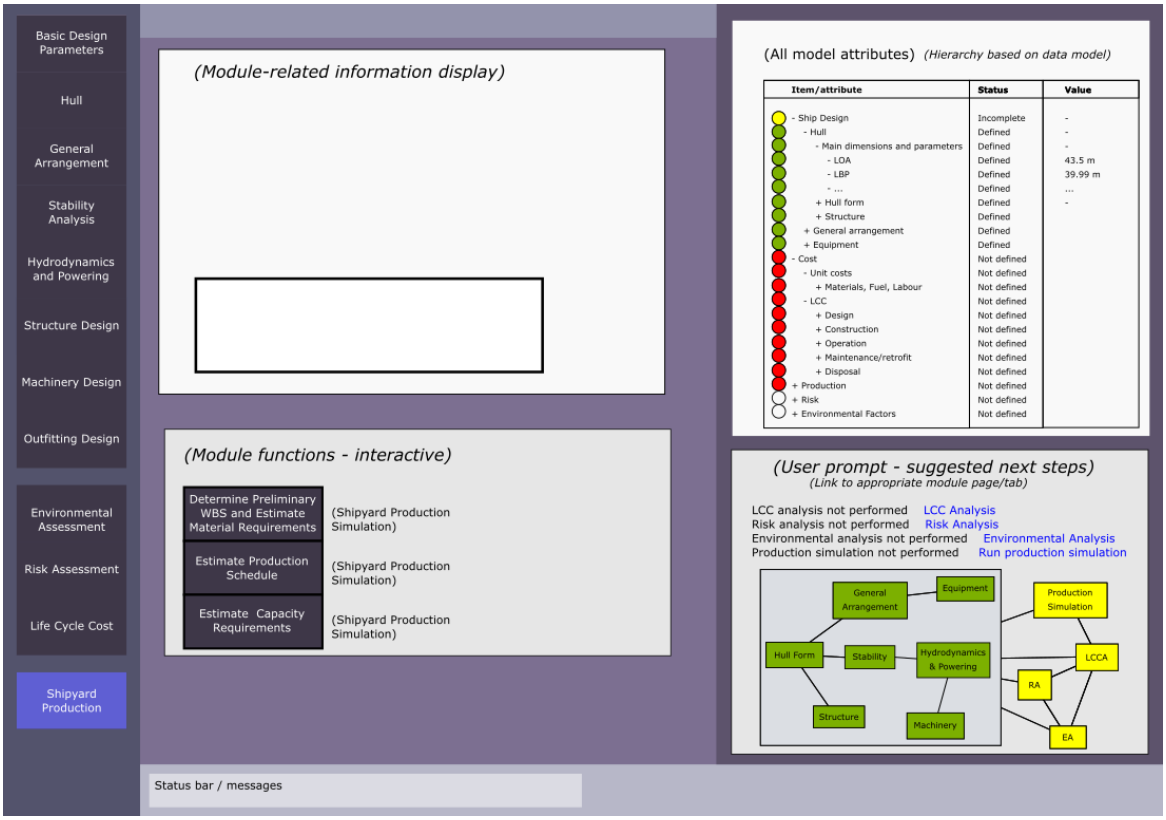

Figure 13: Production analysis module

Once the ship design has been defined to a sufficient degree, the user can start using the modules relating to LCC, Risk, and Environmental Analysis and Production Simulation. The mock interface shows the AES Shipyard Production Simulation software [6] for this function (shown in Figure 13 above) and the IST spreadsheet tool for estimating LCC (Figure 14).

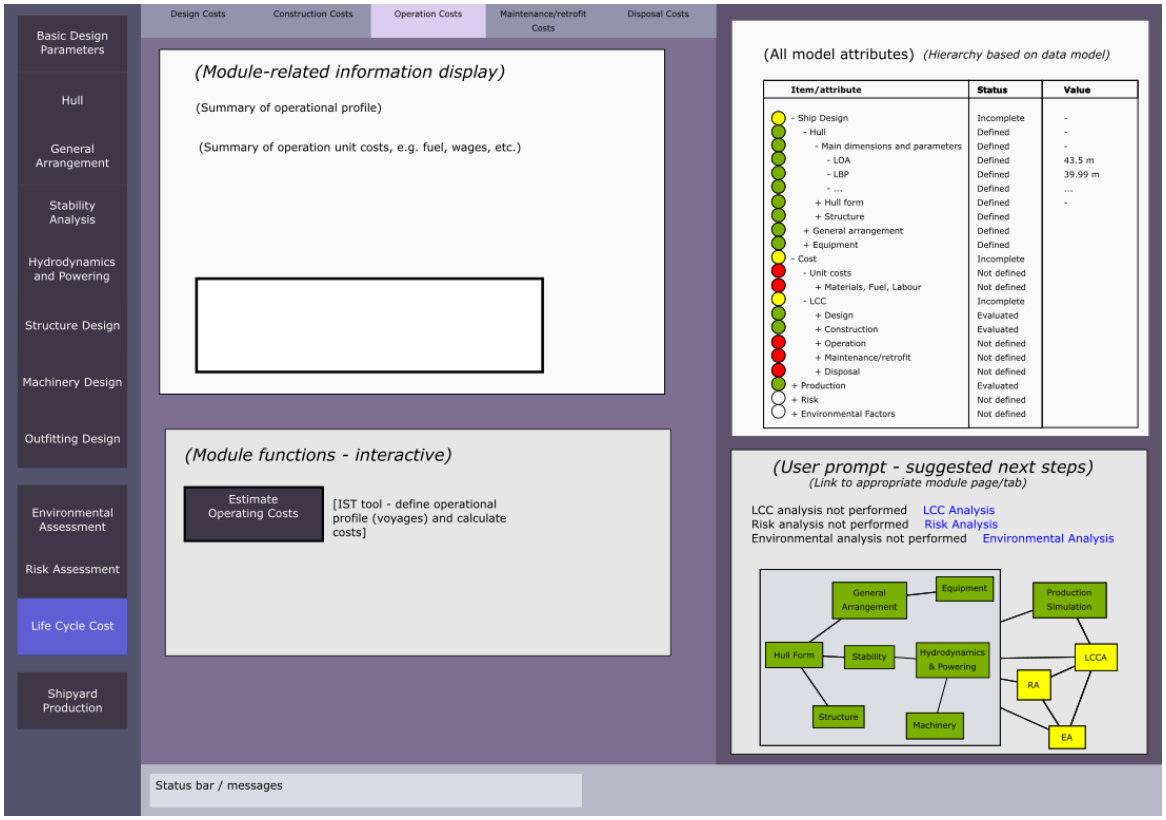

Figure 14: Life Cycle Cost module

3 SHIPLYS Scenario Requirements for integration of rapid virtual prototyping and life cycle tools.

The Consortium have developed a spread sheet based matrix in order to consider tools and components to be integrated the SHIPLYS platform. The matrix shown in figure 15 is envisaged to be a live document to be updated when required together with the Software Integration Register described in section 4.

No	Functionality	Sub-functionality	Activity Diagram (for use in other worksheets)	Existing s/w or approach	Is an alternative s/w or approach that you would like to use ?	To be developed within SHIPLYS	Required for SHIPLYS LCCA Module ?	Required for SHIPLYS LCA Module ?	Required for SHIPLYS Risk Module	SHIPLYS Scenario clearly showing application of the functionality S1 (Y/N)	SHIPLYS Scenario clearly showing application of the functionality S2 (Y/N)	SHIPLYS Scenario clearly showing application of the functionality S3 (Y/N)
A121	EVALUATE REQUEST & SCHEDULE BID	*NOT INCLUDED IN SHIPLYS TOOL*										
A122 (Lead by NTUA)	CREATE PRELIMINARY DESIGN	Create preliminary hull form	A1221	See A122 worksheet			N	N	Y	M	Y	Y (for large works of retrofitting and conversions)
		Create preliminary general arrangment	A1222	See A122 worksheet			N	N	M	M	Y	Y (for large works of retrofitting and conversions)
		Estimate hydrodynamics and power	A1223	See A122 worksheet	subcontract CFD work		Y	Y	M	Y	Y	Y (for large works of retrofitting and conversions)
		Create preliminary structural design	A1224	See A122 worksheet			N	N	Y	N	Y	N
		Create preliminary machinery design	A1225	See A122 worksheet			N	N	N	Y	Y	N
		Create preliminary outfitting design	A1226	See A122 worksheet			N	N	N	M	Y	Y (for large works of retrofitting and conversions)
A123	DECIDE POST-SALES AND MAINTENANCE SUPPORT	*NOT INCLUDED IN SHIPLYS TOOL*										
A124 (Lead by AS2CON + SU)	CALCULATE COST OF SHIP	Calculate cost of design	A1241			Y				N	Y	Y (for large works of retrofitting and conversions. These data should be included in SHIPLYS software)
		Calculate cost of construction	A1242	In-house speadsheet/ST-tool		Y				Y	Y	Y
		Calculate cost of operation	A1243	In-house speadsheet/ST-tool		Y				Y	Y	Y
		Calculate cost of maintenance/retrofitting/risk	A1244	In-house speadsheet/ST-tool		Y				Y	Y	Y
		Calculate cost of scrapping	A1245			Y				Y	Y	Y
A125	PRESENT OFFER *(former A123)*	*NOT INCLUDED IN SHIPLYS TOOL*										
A126 (Lead by SOERMAR)	CREATE PRELIMINARY DESIGN FOR RETROFITTING PURPOSES	Create preliminary machinery and outfitting design via scanning 3-D	A1261	3D scanning Technology (FARO)	Y	N	N	N	N	N	N	Y (some of these data should be included in SHIPLYS software)
		Create preliminary machinery and outfitting design via 2-D drawings	A1262	2D model - AUTOCAD	Y	N	N	N	N	N	N	Y (some of these data should be included in SHIPLYS software)
		Create preliminary machinery and outfitting design via 3-D drawings	A1263	3D model-AUTOCAD/SOLIDWORKS/ FORAN (FDERN, FSYSD, FPIPE, ISOM, FBUILDS, FDESIGN) /CAFE	Y	N	Y	N	N	N	N	Y (some of these data should be included in SHIPLYS software)
A127 (lead by SU)	ESTIMATION OF ENVIRONMENTAL IMPACT	Estimate environmental impact of construction	A1271	In-house speadsheet/Gabi								
		Estimate environmental impact of operation	A1272	In-house speadsheet/Gabi		Y				Y	Y	N
		Estimate environmental impact of maintenance	A1273	In-house speadsheet/Gabi		Y				Y	Y	N
		Estimate environmental impact of retrofitting	A1274	In-house speadsheet/Gabi								
		Estimate environmental impact of scrapping	A1275	In-house speadsheet/Gabi								
A128 (Lead by IST)	ESTIMATION OF RISK	Identify hazard	A1281	In-house speadsheet/software	N	Y	N	N	Y	Y	Y	Y
		Assess risk	A1282	In-house speadsheet/software	N	Y	N	N	Y	Y(B)	Y(A and C)	Y(B and D)
		Estimate risk control options	A1283	In-house speadsheet/software	N	Y	N	N	Y	Y(B)	Y(A and C)	Y(B and D)
		Assess cost benefit	A1284	In-house speadsheet/software	N	Y	N	N	Y	Y(B)	Y(A and C)	Y(B and D)
		Recommend and make decision	A1285	In-house speadsheet/software	N	Y	N	N	Y	Y(B)	Y(A and C)	Y(B and D)
A129 (Lead by AES)	PERFORM PRELIMINARY PLANNING OF PRODUCTION	Determine preliminary work breakdown structure and estimate material requirements	A1291	See A129 worksheet	Y	Y	N	N	N	Y	Y	Y
		Estimate production schedule	A1292	See A129 worksheet	Y	Y	N	N	N	Y	Y	Y
		Estimate capacity requirements	A1293	See A129 worksheet	Y	Y	N	N	N	Y	Y	Y

Figure 15: The SHIPLYS Master Matrix to consider potential Applications to be integrated on the SHIPLYS platform

The sub-sections below summarise the three Scenarios that are being used in the development of SHIPLYS functionalities.

3.1 Scenario 1: Optimised design of a novel hybrid propulsion system in a short-route ferry

This Scenario is aimed at developing software capability to optimize a novel hybrid propulsion system used in a short-route ferry.

The hybrid propulsion system being considered combines internal combustion engines and battery cells. The optimization here is to determine the most suitable combination of propulsion (i.e. the proportion of propulsion directly from combustion and that from battery), using a life cycle approach covering LCC, risk assessments and environmental impacts.

The approach will cover operation and maintenance, scrapping and recycling stages. Potentially, the implications of optimizing the propulsion system on the design and production of the short route ferry will also be considered using the generic functionality of the suite of software created.

More information on the Scenario is available from [7].

3.1 Scenario 2: Development of conceptual ship design with inputs from Risk-based Life Cycle Assessments

This Scenario is aimed at developing software capability to generate conceptual ship design with inputs from risk based life cycle assessments. The scenario is organized in three consecutive tasks and two parallel tasks: Task 1 on conceptual ship design, Task 2 on risk-based structural assessment and Task 3 on risk-based maintenance. Task 4 is focused on fast hull geometry prototyping and Task 5 is related to production assessment. Tasks 4 and 5 will be carried out in parallel with Task 1.

More information on this Scenario is available from [8].

3.2 Scenario 3: Development of software to support early planning and costing of ship retrofitting accounting for life cycle costs and risk assessments

This Scenario is aimed at developing software to support early (bid-stage) planning and costing of ship retrofitting, taking account of LCC and risk management of processes involved. Risk management in the context of this Scenario includes hazard management and project management risks such as scheduling conflicts and the impact of delays. The ship retrofit process is a reengineering process of the vessel which in many cases can involve fundamental changes in the architecture, functionality or operation of the vessel, but the nature of repair and retrofitting projects differs substantially from long term new building projects.

More information on this Scenario is available from [9].

4 Software Integration Register

SHIPLYS will maintain a formal register of the software items that we will integrate into the SHIPLYS overall toolset. This will be a subset of the software register developed elsewhere but with added detail about the entries that is necessary for integration and subsequent possible commercial use. The register will be updated throughout the project lifetime.

1. Name of software.
2. Category of software (e.g. RVP, LCCA, workflow planner, data repository, regulatory resource, other early stage model).
3. Software status characterisation (new development, prototype, partner's background commercial software, 3rd party open source, 3rd party commercial).
4. SHIPLYS model function the software supports.
5. Integration approach needed (Glue code, DAI, API, batch file, etc.).
6. Execution style (synchronous or asynchronous), does it support full automation?
7. Information security classification.
8. Need for SLA (including cost and service level, foreground generation, considerations, etc.).
9. Technical condition assessment, to guide duration and effort of integration activity and including aspects like: compliance with standards, security mechanisms, performance, scalability and availability. Will the s/w/ impose constraints on the execution characteristics of a larger system or not? (bottleneck, data volumes etc.). Parts of this assessment will necessarily be subjective.
10. Future business value assessment, what measurable benefits might this provide to SHIPLYS.
11. Which scenario, if any, does the tool support?
12. What data sources (manual entry, other s/w, h/w) and formats does the s/w support?
13. What output formats does the s/w generate?
14. What is the nature of the HCI and learning curve needed for this tool? (i.e. is it a spreadsheet with no HCI or embedded help or is it a fully supported s/w suite with advanced GUI etc.?)
15. Updates - when selecting a candidate for integration, we must also be mindful of version control for the component and any formal mechanism it uses for updating itself (which could easily break an integrated tool)

5 Conclusions

In its first year, SHIPLYS has settled on an integrative approach for coupling ad-hoc software components and that is described in this deliverable. A variety of ways in which software (of all standards and maturity) can be made to interact is captured by this approach. By considering both the design goals and what the associated data model must be, we outlined both a schema for interaction and a set of services that control data flow itself. Where interactivity is desired between commercial tools and SHIPLYS then we are confident that as long as the prior tool can be made to output content then we can capture that and re-use as necessary.

As part of introductory work that will lead into later work packages, we have also generated a tentative design for the main HCI of the SHIPLYS solution. It is hoped to be intuitive and effective and familiar, being based on the sequential actions found in the relevant ISO standards.

6 References

[1] Fielding, R. T. (2000) www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf

[2] Fielding, R. T. (2008) - http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[3] Hardt, D. (2010) https://tools.ietf.org/html/

[4] BVB Ltd, as2con-Alveus Ltd (2017) http://www.bvbcafe.com

[5] Lloyd's Register (2014) http://www.lr.org/en/services/software/rulescalc.aspx

[6] Atlantec Enterprise Solutions GmbH (2017) http://www.atlantec-es.com/shipyard-production-simulation.html

[7] Wang H, Oguz E, Jeong B, Zhou P. Optimisation of Operational Modes of Short-Route Hybrid Ferry: A Life Cycle Assessment Case Study. International Maritime Association of the Mediterranean (IMAM) Conference; Lisbon, Portugal 2017.

[8] Garbatov Y, Ventura C, Guedes Soares P, Georgiev T, Koch T, Atanasova I. Framework for conceptual ship design accounting for risk-based life cycle assessment (under review for publication). IMAM 2017 (International Maritime Association of the Mediterrean); Lisbon, Portugal 2017.

[9] Porras A, Herrera L, Carneros A, Zanon JI. LifeCycle and virtual prototyping requirements for ship Repair Projects (under review for publication). IMAM 2017 (International Maritime Association of the Meditteranean); Lisbon, Portugal 2017.