# An Open and Collaborative Object-Oriented Taxonomy for Simulation of Marine Operations

**Ícaro A. Fonseca**, NTNU, Alesund/Norway, icaroaragao@outlook.com
**Henrique M. Gaspar**, NTNU, Alesund/Norway, henrique.gaspar@ntnu.no
**Christopher F. Ryan**, UCL, London/UK, christopher.ryan.15@ucl.ac.uk
**Giles A. Thomas**, UCL, London/UK, giles.thomas@ucl.ac.uk

**Abstract**

*This paper introduces a taxonomy for marine simulations, based on entities, states and processes. The taxonomy supports the open source Vessel.js library, which contains empirical models for different vessel analyses following an object-oriented approach: seakeeping, resistance and propulsion system working condition. The proposed taxonomy is applied to a ship design case, having entities as the physical element (e.g. ship), states as the behaviour of the ship for a given condition in time and processes as the combination of various states over time. Two preliminary fuel consumption simulations applying the proposed taxonomy are presented in a design space with different ship and propulsion system options: first the vessel sailing through a route, second through the sea states of an operating region. Results are explored with visualizations and Pareto frontier plots for different design objectives. As the Vessel.js library is intended to become a focus for collaborative work, suggestions for its extension and improvement are also presented.*

## 1. Taxonomy for Marine Simulations

The application of simulations to marine operations has lately been expanding and diversifying in the marine industry. This diversification leads to a great variety of methods and approaches to tackle different simulation problems through the value-chain. However, simulation packages are often application specific, not using common vessel models, input or output file formats. Furthermore, they are often built with proprietary software, which means that even if someone is willing to adapt the code to their own needs, it is not possible to do so.

Even simulation applications written for academic purposes are often not built for reusability or interconnection with other frameworks, even if their source code is publicly available. They may rely on paid numeric packages (e.g. Matlab), and/or they only provide the source code as a printed appendix with no further documentation. If a user decides to simply run the code in their own machine, they may need to copy the content to source files, pay, download and install the correct version of the required package; and only then they will be able to run the scripts. It may be even more difficult to adapt the code to different use cases, or to extract some functionalities for usage in different projects.

Given these problems, this work introduces a simulation taxonomy intended to handle and connect different simulations of marine operations. It uses one physical definition of the vessel to perform simulations of behaviour in operation. An object-oriented, open-source library accompanies the taxonomy. The library is developed in JavaScript, allowing online deployment as web applications with graphical user interfaces, *Gaspar (2017)*.

The library starts with simulation models based on semi-empirical formulas and closed-form expressions. It is intended to be collaborative, so besides having its source code freely available, the library is maintained in an online repository with related documentation. This means that anyone who wants to contribute can discuss, modify or improve it. Over time, this approach should allow easier incorporation of more sophisticated analyses. A motion simulation, for example, could be gradually improved to include strip-theory or panel methods, then time-domain simulation of wave response. The object-oriented approach may also aid the reutilization of code, as the source is split into several objects with different purposes and capabilities.

## 2. Virtual Prototype Model for Marine Simulations

The basis of the taxonomy is the virtual prototype (VP) representation model proposed by *He et al. (2014)* and previously used in a ship design context by *Fonseca and Gaspar (2015)*. The VP representation model is subdivided into three sub models: an entity model (EM), a state model (SM) and a process model (PM), each one defined as follows:

- EM: the entity model defines the physical product or system which is being simulated. It includes design and specification data, 2D and 3D models of the product, and can represent a vessel or other simulated marine system, such as an oil rig or wind turbine, with any level of detailing. For example, general arrangement, structural definition, 3D models and component specifications can be included in the entity model.
- SM: the state model represents the system subjected to external static constraints in any kind of simulation. In a marine operation VP, the SM could be the resistance of a vessel at a certain speed, particular given load, and environmental conditions. The considered states are dependent on the behaviour the VP model aims to simulate, so for example, a hull strength simulation would require different states when compared to a seakeeping simulation.
- PM: the process model is a succession of different SMs to represent system behaviour over time, or, alternatively, it is the EM subjected to a dynamic constraint. A PM could be, for instance, the resistance model of a vessel sailing on a certain mission through different sea states, or the motion response model of a vessel while performing a heavy lifting operation.

Fig.1 illustrates the relationships between the three models with some examples of possible simulations in a marine operations context. The EM supports both other models, as it represents the simulated system. The SM is the EM under a static constraint, and the PM is the accumulation of several SMs. The PM can be decomposed into several SMs or obtained by applying dynamic constraints to the EM. In the design process, both the SM and PM are used as feedback to modify the EM, i.e., according to the results obtained from the static or dynamic models, the designer adjusts the product to improve its performance, for example, optimizing the hull form.

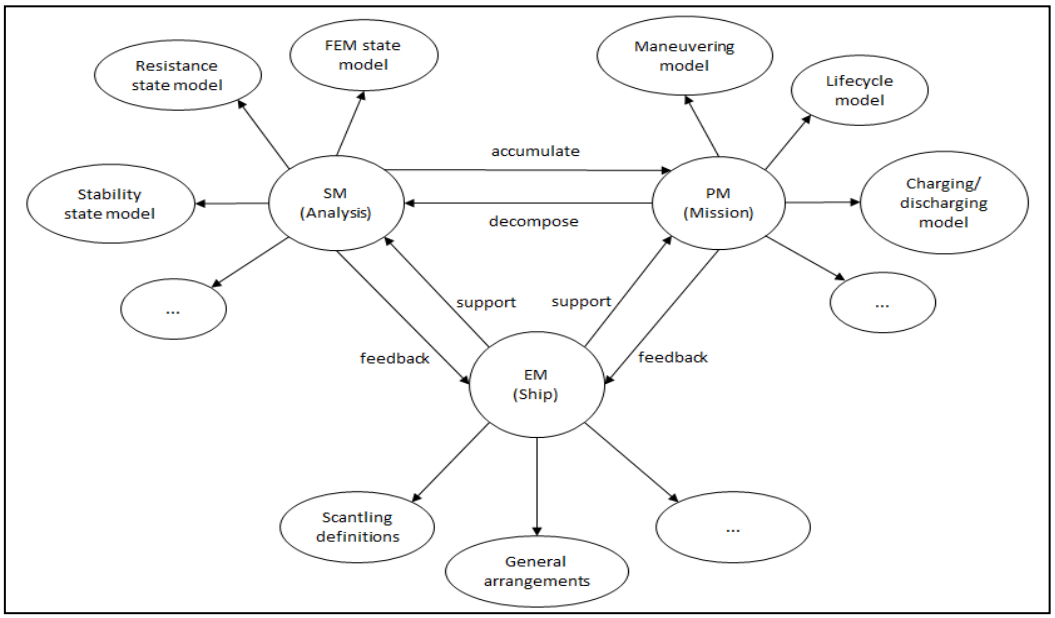


Fig.1: Virtual Prototyping Representation Model Applied to Ship Design as adapted by *Fonseca and Gaspar (2015)* from *He et al. (2014)*

Due to the complexity of a ship, it is natural that a breakdown is performed so as to better handle it as an entity model. In ship design, some traditional approaches to that breakdown are by functional purpose or physical division of systems, for example:

- The System Based ship design methodology, *Levander (2012)*, divides a ship into functional sub systems for estimating the required area and volumes during the initial design stage.
- The Design Building Block approach, *Andrews and Dicks (1997)*, divides a vessel initially into functional blocks containing geometric and technical attributes, which can be further detailed as the design process advances. The authors draw a comparison with the UCL Ship Weight breakdown, essentially a physical division of ship systems.
- The SFI Coding and Classification system (named after the Skipsteknisk Forskningsinstitutt, Ship Research Institute), *Xantic (2001)*, defines a hierarchical ship division based on physical systems arrangement, even if the original definition of those systems is by functional purpose.

The idea of an open taxonomy requires that the EM is able to exhibit and aid handling of both physical and functional characterizations of ship systems. This flexibility yields higher potential for using the models in different contexts during the vessel's lifecycle, i.e., whether the focus is on evaluation of functional performance during conceptual design stage or physical detailing of simulated systems during construction and operation.

State models are snapshots of ship behaviour when subjected to internal or external stimuli, *Gaspar et al. (2012)*. Internal stimuli are simulation constraints which depend solely on states of the internal vessel sub systems. For example, assuming the hull definition of a vessel is known, one can derive its hydrostatic characteristics. Adding weight and spatial definition of internal systems to that model, it is possible to calculate static stability or still-water structural demand for different states of those internal stimuli. The spatial definition, given as a general arrangement or 3D model, could also support a simulation of evacuation time.

External stimuli are simulation constraints which depend on external or environmental states in general and interact with the vessel as a system. Examples of external stimuli are waves, winds, currents or geographic topology. Including these in the model makes it possible to perform simulations such as sailing resistance, propulsion working condition, wave loads and seakeeping.

Finally, process models simulate state changes through time, for example, changes of draft, speed, weather conditions or combinations of these. A simulation of a marine operation is assembled from a sequence of states that change over time. It can be used as a case study to evaluate a vessel's performance in a certain scenario, incorporating external stimuli and operational rules. A lifting simulation, for example, may have an operational rule for interrupting the operation in case the vessel reaches a certain operability threshold according to the current weather state.

## 3. Taxonomy with *Vessel.js*: An Object-Oriented Open Source Library

This section introduces *Vessel.js* as an object-oriented JavaScript library for the taxonomy. Vessel.js is an open source, collaborative project and can be accessed on http://www.vesseljs.org/. The repository includes the latest version of the code, documentation with API reference for users and some examples with tutorials presenting the library's main functionalities.

This work starts from the library version presented by *Gaspar (2018)*. Fig.2 gives a summary of the repository's structure at that time. Folder *classes* includes scripts to define a *Ship* object instance, which represents the entity model and calculates state models which depend solely on internal stimuli. Folder fileIO contains functions to browse, upload and download JSON specification files. Folder *math* contains mathematical operators used to perform calculations.

We expand on this version of *Vessel.js* to perform simulations of fuel consumption with semi-empirical formulas and closed-form expressions. The following sub-sections detail the VP sub models under this framework, including interaction with external inputs and rules, such as sea state definition and vessel operability criteria.

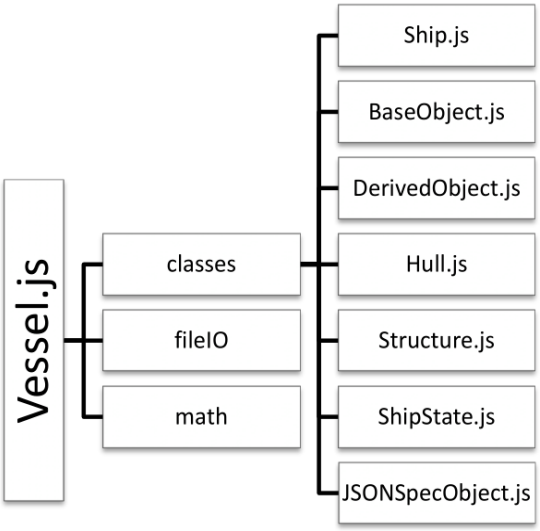


Fig.2: Vessel.js repository structure as of *Gaspar (2018)*. This work expands on the library to perform simulations following the taxonomy.

### 3.1. Entity Model: Ship, Propulsion System

Vessel.js defines a Ship object constructor to create an entity model. The vision for the Ship prototype is to support handling and evaluation of ship design data with different levels of detailing. The object-oriented approach is suitable for handling both physical and functional characterization of ships; for example, it is possible to have objects for ship sub systems and append SFI tags to them as properties.

A 'Ship' object is instantiated from a JSON input specification defining its physical characteristics. A 'Ship' instance includes sub-objects: attributes, baseObjects, derivedObjects, structure and designState, each one with its own set of sub-objects, methods and properties. The first four objects define the EM, while designState handles simulation states.

The 'attributes' object can be used to store general information about the vessel such as descriptions and specifications. The base objects define templates for vessel compartments, tanks and outfitting components. The definition includes spatial dimensions, links to 3D model files, and weight information. Derived objects are the vessel components themselves. They are defined from base objects and a single base object can be used as a template for multiple derived objects. A derived object also adds information in addition to its base object: 3D coordinates describing the component's relative position to the vessel and identification tags according to the SFI group system. The structure object includes bulkheads, decks and hull definition, including its table of offsets. Currently, the most detailed specification available in the Vessel.js repository is a PSV composed of 106 derived objects, as shown in Fig.3. This is the specification used throughout this work.

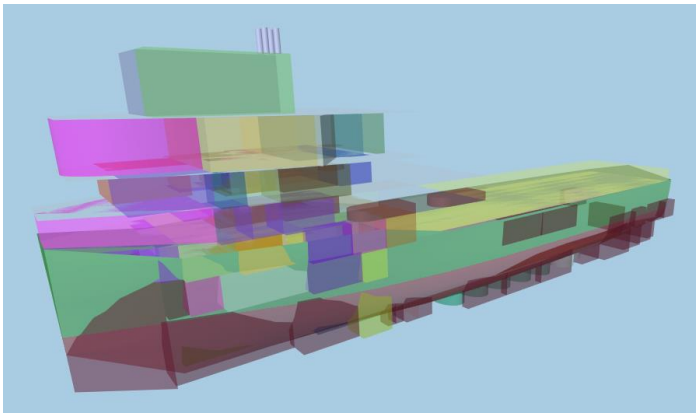

Fig.3: Visualization of an entity model, a PSV composed of 106 derived objects, *Gaspar (2018)*

The entity model was expanded to provide the physical definition required for a fuel consumption simulation. The 'Ship' constructor received a property to specify one or more fuel tanks.

The main PSV specification collects some parameters which are not possible to obtain with the current state of Vessel.js library, i.e.:

- Hull description which is beyond what the library can derive from the table of offsets, for instance whether the hull has a bulb, transom stern or its afterbody form.
- List of hull appendices with respective wetted areas.
- Spare calculation parameters which can be taken as constant, such as natural roll period.

A propulsion system model was also created for the simulations. It is composed of two distinct, independent objects: one for the propellers and another for the power plants.

Propeller objects define physical propeller characteristics: number of propellers, number of blades, diameter and expanded area ratio. Efficiency data is linearized from $K_t$, $K_q$ curves for the expected working regime (i.e., the high efficiency region of the curves). The linearization coefficients are included as properties of the propeller object.

Power plant objects can be assembled from an engine library. For fuel consumption simulations, properties of an engine specification should include MCR and coefficients for approximation of the SFOC curve as a 2nd or 3rd order polynomial.

A power plant object should have at least one main power providing system, as in this diesel electric example:

```
var powerPlant1 = {
    main: {
        etas: 0.95, // shaft efficiency
        etag: 0.95, // generator efficiency
        engines: [CAT_3516C, CAT_3516C, CAT_C32, CAT_C32]
    }
};
```

In this case, the diesel electrical system supplies power for both propulsion and auxiliary systems. Grouping multiple engines in the same array means they are able to share loads. The array order defines the starting order of engines as the demanded power load increases. Shaft efficiency (etas) and generator efficiency (etag) are taken as constant.

A power plant object may instead have one main power plant with two independent systems which do not share loads. For example, two diesel mechanical systems, each with one or more engines coupled to a propeller. In this case, the power plant should have an auxiliary diesel electrical system for powering other systems. In the following example, this auxiliary system includes two high-speed engines:

```
// create a diesel mechanical power plant
var powerPlant2 = {
    main: {
        noSys: 2,
        etas: 0.99,
        engines: [CAT_3516C]
    },
    auxiliary: {
        etas: 0.95,
        etag: 0.95,
        engines: [CAT_C32, CAT_C32]
    }
};
```

Regarding integration between vessel and propulsion system, we favour a modular approach where propeller, power plant and ship specifications are stored in separate libraries, so it is possible to couple and test different combinations during the conceptual design stage (more details are given in section 4.1.). However, this approach does not necessarily exclude a more integrated one. In the future, it would also be possible to define a default propulsion system inside the main ship specification and override it with a different one in case the user wants to test different configurations.

### 3.2. State Model

All calculated states in a state model are handled inside a 'ShipState' object. It originally contained analysis parameters, such as design speed and number of crew in sub-object calculationParameters and a cache with states of ship derived objects in objectCache. Derived object states are purely dependent on internal stimuli, they contain spatial positioning and tank filling ratios with flags for state version control. This work expands the state object to handle ship states which relate to the vessel as a whole system or depend also on external stimuli. These are now stored in a shipCache property, which lists vessel states for one given time instant with a corresponding version flag.

Following the taxonomy, Vessel.js should be able to handle states that depend on both internal and external stimuli. State models which depend purely on internal stimuli are calculated with methods embedded into a 'Ship' instance. In this sense, a 'Ship' instance already includes methods for the calculation of hull dimensions, hydrostatic and stability coefficients based on the current state of its derived objects. Furthermore, a 'Hull' instance includes methods for calculating hydrostatic coefficients as a function of the draft alone. The 'Ship' prototype only needed to be expanded with methods for fuel handling: one for calculating the available amount inside tanks and a second for subtracting the consumed amount.

SMs dependent on external stimuli are obtained using a library of specialized modules which couple to a 'Ship' instance and access it to perform calculations. A simulation of fuel consumption includes modules for wave motion response, resistance, propeller interaction and engine fuel consumption:

- WaveMotion: wave motion response and bending moment due to waves are calculated using closed-form expressions from *Jensen et al. (2003)*. They estimate vessel RAO response as functions of main dimensions, form coefficients and other parameters available in the early design stage.
- HullResistance: the calm water component of hull resistance is calculated based on the *Holtrop (1984)* method. Added wave resistance is calculated with a formula given by Kreitner for wave heights up to 2 m, as suggested by *ITTC (2005)*. For higher waves, a 20% wave margin is added to calm water resistance. This is an analogous solution to that proposed by *Bakke and Tenfjord (2017)*.
- PropellerInteraction: propeller working condition is modelled as discussed in 3.1.
- FuelConsumption: power system working condition is modelled as discussed in 3.1. The fuel consumption state module includes an algorithm for sharing loads among power sources coupled to the same system. The underlying principle is that an additional power source is activated when the previous one reaches 80% of its maximum capacity. This algorithm is heavily based on work by *Voldnes (2017)*.

A 'Ship' object instance received as argument by a state module constructor is linked to a property inside the constructed object, which allows modules' methods to access and modify ship states in an encapsulated manner, as explained in Fig.4. States are modified with method writeResults. A specific array, defined as a property for each state module lists which results should be written to shipState. It can be edited according to the desired output. A memorisation pattern is used to store the results when they are calculated. After stored, they will only be recalculated if the input ship or wave states changed since the previous invoke.

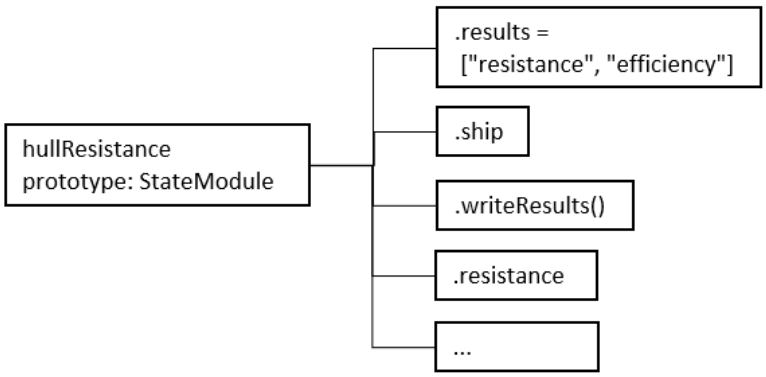


Fig.4: Some of the hullResistance module properties. Method writeResults reads array results and updates the shipState inside the 'ship' property.

When calculating ship states during a simulation, it is necessary to reinforce a specific saving order to assure they are updated correctly, as illustrated by Fig.5. A propeller interaction module requires hull resistance and hull efficiency outputs to calculate the required shaft power, and a fuel consumption module requires the shaft power to calculate a corresponding fuel consumption rate.

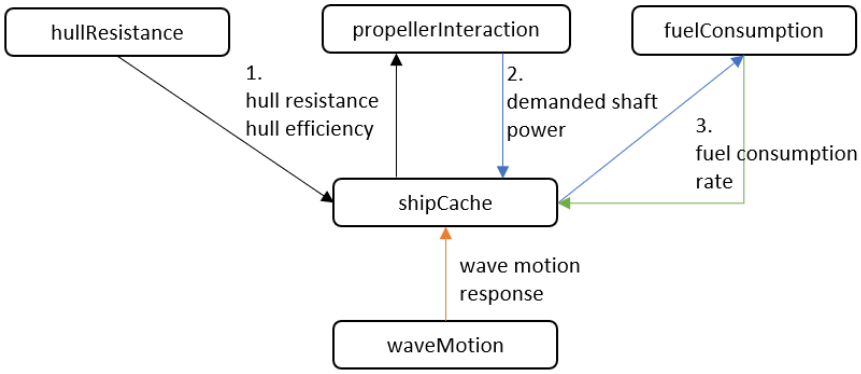


Fig.5: Interdependence between state modules. State calculation and saving sequence should follow numbering to assure results are properly updated. Module *waveMotion* can be used independently.

### 3.3. Process Model

A process model is handled by a master script which assembles and iterates over state modules and external conditions to calculate ship states through time. The script may also evaluate and apply operability rules based on the calculated ship states. After the instantiation of the state modules, the script iterates over them to advance the simulation until a final condition, say, maximum time or sailing destination, is reached. A 'Positioning' object was created to aid calculation of sailed distance, however, it does not follow the template of other SMs, as its depends on duration of iteration time steps and the other modules do not.

Given the modular nature of the state calculations, it is possible to handle different parts of a simulation independently. The same wave motion module, for example, can be used to calculate maximum bending moment caused by incident waves, or to calculate the amplitude of vertical motion response and then use it to trigger an operability rule.

It is also possible to assemble state modules to update different parts of the simulation at different rates. For instance, one could be interested in calculating motion amplitude response every few seconds but would want to update sailing draft due to fuel consumption only every few hours. Some process models illustrating those use cases will be introduced in 4.2.

The final simulation results are a series of states, collected by time step or other label, depending on the simulation. It contains calculated ship states at the very least but can be expanded to include states of derived objects, such as fuel tanks, if relevant.

### 3.4. A Simple Simulation Example

This section exemplifies a simulation with a simple process model example where the ship travels through a path while updating fuel tank level and draft, as shown in the flowchart in Fig.6. The code was simplified for easier comprehension.

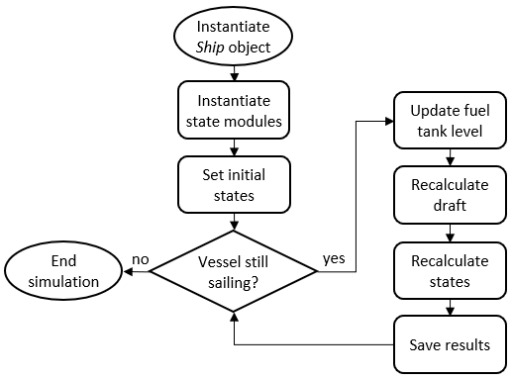


Fig.6: Flowchart of the fuel consumption simulation

A process model script starts with instantiation of a ship object:

```
var myShip = new Vessel.Ship(shipSpec);
```

The 'Ship' object handles the EM and performs SM analyses which are not dependant on external stimuli. It also stores current vessel states. State models which interact with external stimuli, in this case wave condition, are calculated with additional modules. In the example below, the modules are instantiated and the initial states are written to the vessel's state cache following the update sequence in Fig.5 (resistance, propeller interaction and fuel consumption):

```
var hullRes = new HullResistance(myShip, wagProp, waveDef);
var propInt = new PropellerInteraction(myShip, wagProp);
var fuelCons = new FuelConsumption(myShip);
hullRes.writeResults();
propInt.writeResults();
fuelCons.writeResults();
```

The script will simulate the PM until the vessel reaches its destination. At every iteration, the script calculates the consumed fuel for the last time step, subtracts it from the fuel tanks, calculates a new draft and updates the ship state accordingly.

```
while (stateHistory[time].travelDist < pathDist) {
    consumedFuel = timeStep * shipState.consumptionRate;

    myShip.subtractFuelMass(consumedFuel);

    hullRes.setDraft();

    hullRes.writeResults();
    propInt.writeResults();
    fuelCons.writeResults();

    position.advanceShip(timeStep);

    Object.assign(stateHistory[time], shipState.state);
}
```

In the example above, states are saved to ‘stateHistory’ at each time step. It becomes a record of ship states through the process model, which can later be downloaded as the final simulation result.

## 4. Fuel Consumption Simulations with Vessel.js

The following subsections will demonstrate how different fuel consumption simulations can be performed over multiple designs with Vessel.js. Following the taxonomy, a design library of vessels and propulsion system objects will be created by reusing modified versions of the same physical system templates, or entity models. Two different process models will be assembled using the state modules presented in the previous section.

### 4.1. Design Library: Ship and Propulsion System

The ship library was planned to allow exploration of main dimension combinations during the conceptual design stage. It includes a visualization based on Three.js (https://threejs.org/) which receives a parent ship specification and allows the user to linearly scale it inside a given ratio range for length, beam and depth, as shown in Fig.7. When one dimension is modified, the others are automatically adjusted so as to keep displaced volume constant for a scaled draft. It is also possible to fix one dimension and vary another, for instance, fix length, vary beam, and let depth be automatically adjusted by the app to keep volumes constant.

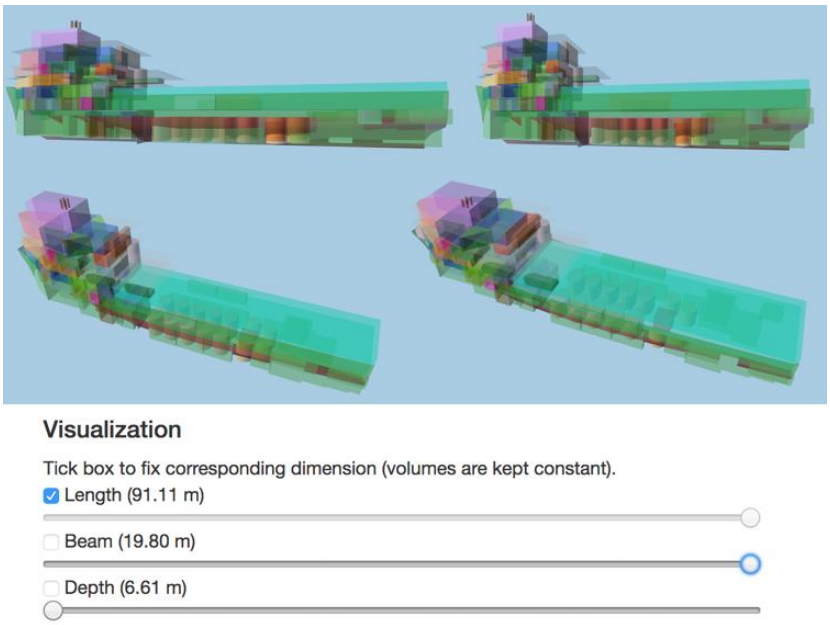


Fig.7: Visualization of different PSV designs created by parametric transformation

The app derives ship object instances based on the design space explored through the visualization. It fixes the beam at a base value, loops through the length, steps over the beam, loops again through the length, and so on. Weights of hull, decks and bulkheads are scaled with their areas. Lightweights of tanks and compartments are assumed to remain constant. As capacities of tanks and compartments scale with volume, they also remain constant. Fixing the scaling ratio range from 0.9 to 1.1 of the parent specification main dimensions and using a scaling step of 0.01, a total of 441 ship instances are generated. Many of them were discarded due to being out of the validity range for the Holtrop method, but valid instances still summed up to 209 for the parent PSV specification used in this work.

The propulsion library contains propeller and power plant objects stored as JSON specifications. The propeller library comprises a selection of Wageningen B-series propellers with typical parameters for ship propulsion (3 to 5 blades, pitch ratio 1.2, expanded area ratio from 0.55 to 0.65).

The power plant library comprises two power plants. The first has two medium speed diesel mechanical systems each linked to one propeller, and two diesel electric systems for auxiliary power. The second power plant has the same four engines as the first one, but all linked as a diesel electric system which shares power loads from propulsion and auxiliary systems. This power plant library was partially adapted from work by *Voldnes (2017)*.

### 4.2. Process Models for Fuel Consumption

In this section we present two process models for different fuel consumption simulations, both relying on the taxonomy and state modules previously introduced. The first model, path, simulates the vessel's fuel consumption sailing through given route and wave conditions. The second model, lifecycle, simulates a yearlong trip through the sea states listed on an input annual scatter diagram. The path model can be used for calibration of the simulation based on previously known data for a given trip or, in the future, for practical estimations of fuel consumption. The lifecycle model can be used for evaluation of design suitability for a given operating region.

#### 4.2.1. Path

The path simulation has different update rates for the verification of a new wave state (10 s), fuel tank levels (1 min) and sailing draft due to fuel consumption (1 hour) by default, which can be modified on the HTML interface. The script also contains a rule to slow down the vessel speed by 10% if a vertical acceleration threshold is exceeded, and then try to return to default sailing speed when the wave state changes. This is intended to simulate a voluntary speed reduction in case of excessive motions.

A method was added to change the wave definition back and forth between two different regular wave states at every hour. In the future, this method could be elaborated to incorporate stochastic representations of wave state.

The script starts with the instantiation of a 'Ship' object from a specification and assignment of departure fuel tank levels. Then, it instantiates state modules and calculates initial states akin to what was shown in 3.4. The script then iterates with a base time step of 1 s. It assigns results from the previous time step to the current one and proceeds to verify if it should update states related to time steps for fuel tank levels, draft, and wave state. If so, then it executes the necessary commands for the update and writes the results down to the state cache. By the end of an iteration loop, the operational rule for the vertical acceleration threshold is reinforced and the ship advances one time step in the sailing course. For each change in vessel or tank state, the results are also updated in the current time step of stateHistory. The simulation continues until the ship reaches its destination or runs out of fuel. Fig.8 presents a simplified flowchart of the model.

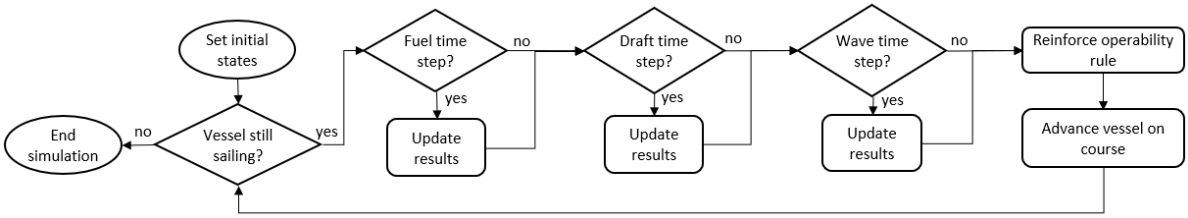


Fig.8: Simplified flowchart of the path process model. A complete flowchart can be consulted in the Vessel.js online repository.

#### 4.2.2. Life Cycle

The lifecycle script simulates the ship traveling through all the wave states listed on the scatter diagram for a geographic region. It assumes an equal share of the yearlong trip time for each sea state occurrence on the diagram.

The algorithm is very similar to that presented previously, but instead of iterating until a destination is reached, it iterates through the sea states. Also, the model does not consider fuel tank level or sailing draft variations, as these are not specific to a given sea state.

For each sea state, the script creates a regular wave matching the given characteristics (peak period and significant wave height). A simulation then starts with the vessel sailing through that wave, if necessary reducing the speed until the acceleration threshold criterion is fulfilled. When the motion criterion is fulfilled, the algorithm uses the ship state to calculate travelled distance and consumed fuel for that sea state. Finally, it calculates total fuel consumption and average speed for a yearlong trip. Fig.9 presents a simplified flowchart of the model.

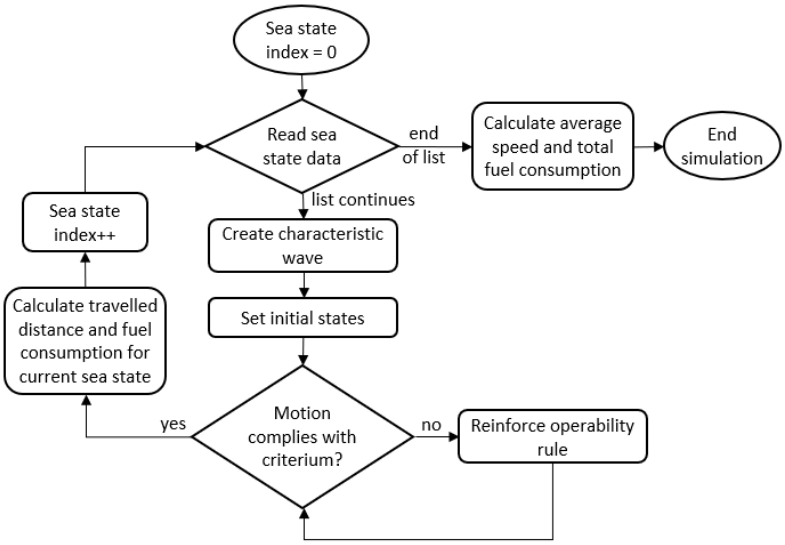


Fig.9: Simplified flowchart of life cycle process model

## 5. Results Exploration

The life cycle process model was simulated for the 209 ship specifications and 2 power plant specifications introduced, with 2 different propeller configuration options, totalling 836 designs. At this stage, we do not focus on quantitative values, as the model still needs to be validated before results are deemed reliable. The focus is on the utilisation of the same taxonomy to perform different types of simulations for different design proposals, and then on exploration of the performance trade space based on the results.

Two different result visualizations were built for this work using the open source JavaScript library D3.js, *Bostock et al. (2011)*: a parallel coordinates plot, Fig.10, and a scatter plot with Pareto frontier, Fig.11. These are used as example, but there are several other ways to visualize and explore ship design information, *Calleya et al. (2016)*.

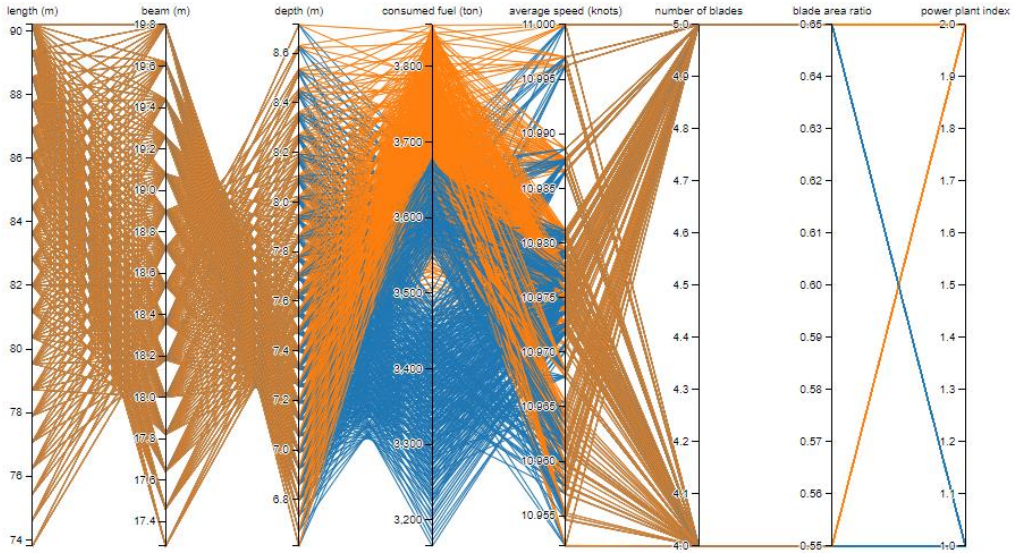


Fig.10: Parallel coordinates plot for a life cycle simulation of 836 diferent designs, colours to differentiate power plants

The parallel coordinates plot has axes for ship main dimensions, fuel consumption, average traveling speed, propeller characteristics and power plant identification number. It has functionalities to support design space exploration, such as reordering of the axes to identify correlation among variables and filtering through selection to answer questions such as “what is the most efficient design among the fastest ones?” or “what is the most efficient design with diesel-electric main propulsion?”. The scatter plot shows fuel consumption and average attainable speed with a corresponding Pareto frontier.

The analyses yielded feasible results: long ships with narrow beams tended to achieve better fuel efficiencies than short ships with wide beams. Naturally, in a design context, stability requirements and roll motions would at some point become compromised when choosing designs with narrower beams. The exploration proposed here can be expanded to tackle those concerns using the Vessel.js library. For example, the metacentric height could be plotted on an additional axis in the parallel coordinates plot, or designs with a GM below a certain threshold could be discarded.

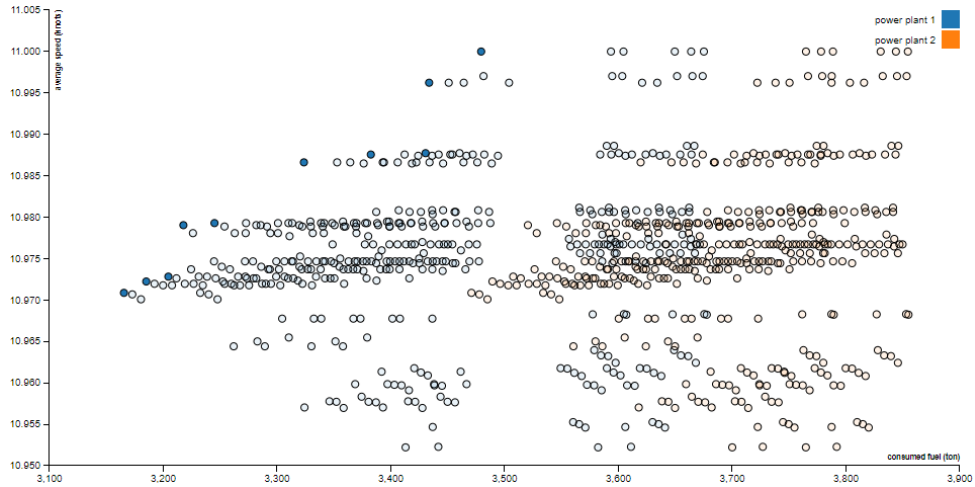


Fig.11: Scatter plot with Pareto frontier for fuel consumption and average ship speed. Dominated solutions appear with opaque filling.

The lifecycle and path apps can be accessed on the project’s repository. The online version of the path app reduces the number of simulated ship instances to some dozens to keep running time in accordance with what is generally expected from a web app. Nevertheless, the path app is still capable of simulating several hundreds of designs when running locally on the client side, but it may take a little longer.

## 6. Guidelines for Extending the Taxonomy and Library

This paper introduces the first step in the simulation of marine operations using the open source Vessel.js library. This is done with a taxonomic framework which divides the simulation in three interdependent models. The entity model is handled with objects for ship and propulsion system. State models which are dependent solely on internal stimuli are calculated with methods embedded into a 'Ship' object instance. State models which also depend on external stimuli are handled with a library of modules which perform calculations using the 'Ship' object. The process model is a script which calculates ship states, wave states and operational rules to perform a simulation.

The simulation models used here are quite simple, but they are created with features to foster collaborative development, such as standardisation, modularisation and, in the future, reference documentation. This means the models can be either modified, adapted for specific use cases, or improved by anyone who wishes to make public contributions.

The next applications should expand the taxonomy to deal with motions and other states which evolve through time. The simulations presented in this work relied mostly on sequences of stationary states. They did not model transitions between consecutive states; even wave motion response was calculated as amplitudes. An improved handling of motions and positioning will open the way to the simulation of manoeuvring and interaction between multiple entity models, for instance, a multibody simulation of two or more vessels keeping station next to each other.

In terms of accuracy, the fuel consumption simulation model should be validated with operational data before it is used for practical applications. One suggestion is to compare it to PSV operational data studied by *Voldnes (2017)*, which could also be used as a reference to expand the simulation to a complete operational profile, rather than simply fuel consumption in transit condition. This would include expansion of the entity model to better represent PSV propulsion, including dynamic positioning capabilities.

Furthermore, the current calculation of traveling speed in waves can be improved by coupling it to an attainable ship speed model such as the one proposed by *Kwon (2008)*.

In terms of computational capabilities, all the simulations presented in this work were performed with web browsers running on client-side. A natural step for the future will be to make the library compatible with a server-side framework, most probably Node.js (https://nodejs.org/). This will allow the simulations to access higher computational power and execute heavier models. A strip theory model, for example could be integrated to the library either as native JavaScript code or as an external open-source module, for example PDSTRIP, *Bertram et al. (2006)*.

**Acknowledgements**

The Vessel.js source code benefited heavily from Elias Hasle's (NTNU, Norway) expertise in programming. This research is connected to the Ship Design and Operations Lab at NTNU in Ålesund, which is partly supported by the EDIS Project, in cooperation with Ulstein International AS (Norway) and the Research Council of Norway. BMT for the support of Mr. Christopher Ryan's PhD studies.

**References**

ANDREWS, D.; DICKS, C. (1997), *The building block design methodology applied to advanced naval ship design*, Int. Marine Design Conference (IMDC), Newcastle

BAKKE, M.Ø.; TENFJORD, P.S. (2017), *Simulation-Based Analysis of Vessel Performance During Sailing-Describing a simulation platform applied in early stage ship design*, Master's Thesis, NTNU, Trondheim

BERTRAM, V.; VEELO, B.; SÖDING, H.; GRAF, K. (2006), *Development of a freely available strip method for seakeeping*, 5th Conf. Computer and IT Applications in the Maritime Industries (COMPIT), Leiden, pp.356-368

BOSTOCK, M.; OGIEVETSKY, V.; HEER, J. (2011), *D³: data-driven documents*, IEEE Trans. Visualization and Computer Graphics 17, pp.2301-2309

CALLEYA, J.; PAWLING, R.; RYAN, C.; GASPAR, H.M (2016), *Using data driven documents (D3) to explore a whole ship model*,11th System of Systems Engineering Conf. (SoSE), pp.1-6

FONSECA, Í.A.; GASPAR, H.M. (2015), *An object-oriented approach for virtual prototyping in conceptual ship design*, 29th European Conference on Modelling and Simulation (ECMS) Conf., Varna, pp.171-177

GASPAR, H.M.; RHODES, D.H.; ROSS, A.M.; ERIKSTAD, S.O. (2012), *Addressing complexity aspects in conceptual ship design: A systems engineering approach*, J. Ship Production and Design 28, pp.145-159

GASPAR, H.M. (2017), *JavaScript Applied to Maritime Design and Engineering*, 16th Conf. Computer and IT Applications in the Maritime Industries (COMPIT), Cardiff, pp.428-443

GASPAR, H.M. (2018), *Vessel.js: an open and collaborative ship design object-oriented library*, Int. Marine Design Conf. (IMDC), Helsinki

HE, B.; WANG, Y.; SONG, W.; TANG, W. (2015), *Design resource management for virtual prototyping in product collaborative design*, Proc. Institution of Mechanical Engineers, Part B: J. Eng. Manufacture 229, pp.2284-2300

HOLTROP, J. (1984), *A statistical re-analysis of resistance and propulsion data*, Int. Shipbuild. Prog. 31, pp.272-276

ITTC (2005), *Full Scale Measurements, Speed and Power Trials, Analysis of Speed/Power Trial Data (7.5-04-01-01.2)*, ITTC - Recommended Procedures and Guidelines.

JENSEN, J.J.; MANSOUR, A.E.; OLSEN, A.S. (2004), *Estimation of ship motions using closed-form expressions*, Ocean Eng. 31, pp.61-85

KWON, Y.J. (2008), *Speed loss due to added resistance in wind and waves*, Nav. Arch. 3, pp.14-16

LEVANDER, K. (2012), *System based ship design*, NTNU Marine Technology, Trondheim

VOLDNES, B. (2017), *Simulation of marine hybrid machinery systems based on vessel operational data*, Master's Thesis, NTNU, Trondheim

XANTIC (2001), *SFI Group System - A system for classification of technical and economic ship information - Product Description*, Xantic, Svendborg