A Public API Supporting Autonomous Navigation

To cite this article: S P Berge et al 2019 J. Phys.: Conf. Ser. 1357 012037

View the article online for updates and enhancements.
A Public API Supporting Autonomous Navigation

S P Berge¹, M Hagaseth¹, T I Bo¹
¹SINTEF Ocean AS, Marinteknisk senter Tyholt, Otto Nielsens vei 10, 7052 Trondheim, Norway.

svein.berge@sintef.no

Abstract. This paper presents a public API proposed for autonomous maritime operations where several moving and fixed objects are involved using wireless communication solutions. The API ensures that high-level data can be exchanged without knowing details of the underlaying implementation. The data will be available for 3rd party vendors to connect their applications and equipment to the proposed framework for data exchange. The high-level data covers information about the object's geometry, the object's uncertainty zone (position uncertainty zone), sensor data, communication solutions and different operational zones (restrictions) dependent on the actual operation and its states. A common data model defining the necessary data, its meaning and their relationships are needed to ensure correct usage of the data across applications and equipment from several vendors. In addition, required functions and services are defined based on the Hull-to-Hull pilot system developed in the Horizon 2020 project "Hull-to-Hull". The paper also presents a chat application that transmitts small messages between several ships by using the defined services and data formats.

1. Introduction

Traditionally, ships are equipped with equipment from different suppliers, and system integration is both time consuming and expensive. Since most ships are specialized with respect to operation and technical solution, there is a large potential to use known and open protocols and interfaces to reduce both time and costs from design, production, operation and approval. In the Open Bridge project [1], focus is to develop both common interfaces and standard graphical components and symbols such that any supplier can connect their equipment to other Open Bridge compatible ship bridge systems. Since most bridge system today uses the ship's local area networks to exchange data, there are several IP protocols already available like TCP and UDP that are available as services. On the application level, protocols like HTTP/1.x and Remote Procedure Calls (RPC) using HTTP/2 will support synchronous communication in a request-response pattern well suited for client-server applications. HTTP is also well suited for listing available services onboard. For time critical real-time applications, these protocols are not necessarily reliable. When the HTTP protocol is used on top of TCP, retransmission of missed data is provided. TCP guarantees the sequence of received packages. However, this means that there is no guarantee for real-time response, as the delay will vary. Most sensors are connected to the network on board using UDP since this protocol has no retransmission of messages, and thus has a fixed delay. However, the UDP does not guarantee that packages are well received within time and order.

For maritime application, we therefore need to know the behaviour of the different protocols in both the application, transport, data link, and physical layer from the OSI-model. When introducing wireless communication between ships, the total system performance will depend on the physical
layer. With respect to maritime safety, the overall performance and reliability seen from application to application on each ship will depend on several factors, like bandwidth and latency over the communication link.

The motivation for this study, is to develop a public API for ship-to-ship communication applied for autonomous navigation where several objects are involved. This framework must consider both requirements supporting autonomous navigation services and services for safe navigation which highly depends on the actual context. The API will support open interfaces for both navigational support and services for monitoring systems' health to avoid critical failures and accidents. The proposed solution will only define the API for data and services, not the implementation. The main objective is therefore to develop an open framework for the Hull to hull (H2H) project covering both API and working process that can be used by any third-party vendor that can connect their equipment and applications directly to the network onboard and use the wireless communication solution between fixed and movable objects to transfer data needed for navigation.

During the development of the framework, we have prototyped a simple chat application, where users can transmit messages from an object to any other objects. We have used gRPC (Google Remote Procedure Calls) as the RPC framework and applied it for maritime applications. The gRPC framework supports the process of first defining data and services, then to derive the open APIs and lastly to develop and test the services.

2. The H2H project
The H2H project [3] is funded by the European GNSS Agency under the Horizon 2020 program. The project is coordinated by Kongsberg Seatex (NO), and participants are SINTEF Ocean (NO), SINTEF Digital (NO), KU Leuven (BE) and Mampaey Offshore Industries (NL). The project started in November 2017 and will run for three years. The project will develop the H2H concept, propose a standardized framework needed to support the concept and study safe and secure communication solutions. A H2H pilot will be built and demonstrated in three use cases; simultaneous operation, inland waterways and auto-mooring.

2.1. The H2H concept
The core functionality of H2H is to provide hull to hull distance between vessels, and to use the concept of uncertainty zone to visualize the uncertainty of the distance calculation.

Both ships must know the location of the hull relative to each other to be able to calculate the distance between them at various points at the hull. In H2H this is obtained by exchange of position sensor data and 2D and 3D geometric vessel models between H2H objects, as shown in Figure 1. The geometric vessel models will be used to generate digital twins representing the geometry of the vessels, and then the sensor data will allow positioning the digital twins relative to each other. Each H2H object is represented by a digital twin holding information about its geometry, the sensor information that the uncertainty zone is calculated based on, and the accuracy of the data.
In addition to hull to hull distances, the hull to hull velocities are essential information for navigation. The system will therefore also estimate the relative motion between the digital twins, and from this derive hull to hull velocities.

The position sensors can be of different types, including systems providing two- and three-dimensional positions (for example GNSS) and systems providing range measurements and angle measurements, as well as inertial systems. In the H2H pilot we will include the European GNSS systems Galileo and EGNOS. Galileo will be used in relative mode providing high accuracy relative positions, whereas EGNOS will provide integrity. The uncertainty zone will be derived based on the accuracy of the positioning sensors and the accuracy of the geometric model. The concept is extended to not only providing hull to hull distances, but also distances between hull and static objects, for example a quay.

As shown in Figure 4, the H2H system has two external interfaces:

1) The H2H Engine User Interface which provides data and services to applications using the positioning functionalities provided through the H2H concept, for instance auto-mooring application, single-handed sailing support applications and applications to support simultaneous operations, for instance ensuring that two ships can operate in a closer proximity and still maintaining the required level of safety.

2) The H2H Vessel-to-vessel Interface ensuring vendor-independent exchange of positioning data, geometry data and accuracy measurements needed for the detailed calculation of relative positioning.

Both interfaces will be based upon existing standards, when available, to enable different vendors to connect their own proprietary applications and systems following the H2H framework.

2.2. The H2H conceptual model
The conceptual model defines the system architecture and facts about what the architecture covers, including the area of interests. Since there are many types of ships with different level of instrumentation and communication capabilities, the domain model needs to include all necessary instances and relationships between these instances, see Figure 2.
To enable consistent exchange of data between the objects, both moveable and fixed, a conceptual data model is defined as described in Figure 3. The main concepts that are defined, covers the following: Operation, Object, Sensor, Communication, Geometry, Uncertainty zone, Operational zone, and Message. The conceptual model represents the "real world" describing interaction of several objects in the context of autonomous navigation. Based on the conceptual model, a subset of the concepts have been implemented as tables in a relational database PostgreSQL [2], with support for spatial and geographic objects.

Figure 2 H2H Conceptual Model

Figure 3 H2H conceptual model
2.3. Interfaces

In principle, both the ship-internal interface (H2H Engine to H2H Application) and the external interface (H2H Engine to H2H Engine) between ships/objects are public APIs, Figure 4. For the external interface, additional data from target ships/objects (e.g. GNSS pseudo-range measurements) will be transmitted in order to compute more accurate position estimates of own position estimate. Target objects are considered as additional sensors that will be used to derive the digital twin of all involved objects in area of interest (both own object and other H2H targets).

For both interfaces we define two main categories of data:

1. Navigational data for real-time monitoring and control (synchronous data) which is time critical for navigation either in open or closed loop applications.
2. Non-real time (asynchronous) data supporting two-ways communication (status, systems health, configuration, man-machine interface)

3. The Open API Initiative (OAI) and Open API specification (OAS)

The Open API Initiative's main objective is to develop a vendor neutral description format used to promote interoperability and reuse of software components. With reference from the website [4]:

"The Open API initiative (OAI) focused on creating, evolving and promoting a vendor neutral description format. APIs form the connecting glue between modern applications. Nearly every application uses APIs to connect with corporate data sources, third party data services or other applications. Creating an open description format for API services that is vendor neutral, portable and open is critical to accelerating the vision of a truly connected world."

The current specification is denoted as the OpenAPI Specification, which defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. When properly defined via Open API, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, the Open API Specification removes guesswork in calling a service.
3.1. REST API

REST (Representational State Transfer) is a software architecture style used for creating web services (REST APIs) that provide interoperability between internet applications. The REST APIs are defined in a client-server architecture, are stateless in that no client context is stored on the server between requests, and important: The architecture ensures decoupling of each part in that individual resources can be fully specified in a client request for instance as a URI, and the server can respond with HTML, XML or JSON independent of its internal implementation. The REST APIs uses the HTTP/1.x protocol which is end-to-end data communication (TCP/IP). Swagger (https://swagger.io/) offers different tools for using the Open API specification file and can create, update and share Open API definitions with other customers. Using the swagger editor and code generation toolbox, code for both servers and clients can be generated automatically. These code stubs can be integrated into different applications like for example web-clients using REST services.

3.2. Remote Procedure Calls (RPC)

Similar to REST API, the Remote Procedure Calls (RPC) allows user defined code to be executed in a different address space (commonly on another computer on a shared network). Typically, the client-server interaction is implemented via a request-response message passing system. Instead of calling a local method, the method is run on another computer which shares data on the network.

![Figure 5 gRPC client and server][5].

The Google Remote Procedure Calls (gRPC) uses the Google Protocol Buffer to optimize the payload on the network and generates necessary stubs for client and server. The advantage of using the gRPC compared to OpenAPI are as follows [6,7]:

- Compared to REST API, the gRPC uses the HTTP/2 instead of HTTP/1.1. The HTTP/2 protocol offers more streaming of data both ways. Both client and server may transmit stream of data backwards and forwards.
- Both the data and services are defined in a proto file and code is automatically generated using the protocol application. The data packages are more compact since it uses the Google
Protocol buffer to optimize the payload with respect to size. This is important in maritime applications with limited bandwidth capacity for the wireless link.

- In addition, there are built-in mechanisms for max response time.
- If the connection is lost, both client and server will be notified, and the application can take the proper action.
- Only one connection for each client.
- Data transfer is binary for HTTP/2 compared to textual for HTTP/1.1.
- Data transfer overhead is less for HTTP/2.
- Faster transfer of large files (e.g., images) since HTTP/2 offers better bidirectional streaming.

4. Chat Application
A simple chat application has been used as first iteration to develop a public API and services to demonstrate usage of gPRC and also REST API. In addition, the automated working process from a simple specification file to runnable software is

4.1. Workflow
Figure 6 shows the Chat.proto file which is the specification file for the chat application (both data and services).

```proto3
syntax = "proto3";

//import "google/protobuf/timestamp.proto";
import "google/api/annotations.proto";
import "google/protobuf/empty.proto";

package no.sintef.ocean.hull2hull.data.chat;

message ChatMessage {
  string from = 1;
  string message = 2;
  int32 timestamp = 3;
  repeated string receivers = 4;
}

message ChatMessageFromServer {
  int32 timestampserver = 1;
  ChatMessage message = 2;
}

message ListAllMessagesFromServer {
  repeated ChatMessageFromServer messageList = 1;
}

service ChatService {
  rpc chatGRPC(stream ChatMessage) returns (stream ChatMessageFromServer);
  rpc chatREST(ChatMessage) returns (ChatMessageFromServer) {
    option (google.api.http) = {
      post: "/v1/chat"
      body: "";
    }
  }
  rpc chatGetAllMessages(google.protobuf.Empty) returns (ListAllMessagesFromServer) {
    option (google.api.http) = {
      get: "/v1/chat/getall"
    }
  }
}
```

**Figure 6** Proto-file definition
Typical workflow will be:

1. As a starting point of the workflow is to define the proto file. This file includes data and services for the chat application. The chat function sends chat messages with time stamp and name of sender to all other clients via the server.
2. The proto file is then used to generate code for both the gRPC server and client.
3. The proto file is then used to generate the Open API specification file. This file is used to generate the REST API code for both server and client.

**Figure 7** Typical workflow for code generation

4.2. Chat application interfaces

**Figure 8** Servers and clients for chat application
Figure 8 shows how clients and servers are connected to two different ships. When communication link is established, a client will establish a link to the others ship's server (light blue colour). Over the communication link, the gRPC will use the HTTP/2 protocol with optimized binary google protocol buffer as the payload. The client on ship #2 will receive all messages from ship #1 and vice versa. If the communication link fails, the clients will terminate on both sides. The system will continue to work as two standalone systems. When communication link is re-established, new clients will connect again. The light blue clients will act as gRPC bridges between the two objects.

4.3. Web chat application

The web chat application, see Figure 9, is developed using Vaadin framework [8] as a template. The framework is an open source web framework. The source code is written in only one programming language Java and the framework offers good looking graphical components. The data communication between web server and frontend (web application) is using WebSocket as communication protocol. WebSocket is distinct from HTTP. Both protocols are located at layer 7 in the OSI model and depend on TCP at layer 4. Although they are different, it is designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies and intermediaries, thus making it compatible with the HTTP protocol. To achieve compatibility, the WebSocket handshake uses the HTTP Upgrade header to change from the HTTP protocol to the WebSocket protocol. The WebSocket is also proposed to be used in the Open Bridge project [1] supporting common web application layouts for graphical components.

![Figure 9. Chat Web Application.](image)

5. Conclusions

This paper described an API suitable for exchanging necessary information between fixed and moveable objects as ships, quays and locks to provide autonomous positioning and navigation support. An implementation of the proposed API can support interoperability between ships and other objects equipped by different 3rd party equipment providers. Starting with the data and service specification, this paper described the working process on how to automatically generate OpenAPI specifications with REST API support. In addition, the process on how to develop gRPC based services, was described.

A simple chat application has been developed in the Vaadin framework using both gRPC and REST API for data storage in the PostgreSQL database. The application will send messages both internally on the ship's local network, but also transmitting data to another object(s).
References

[1] Open Bridge Project, http://openbridge.no/

[2] PostgreSQL database, https://www.postgresql.org/

[3] The Hull-to-hull project web site: https://www.sintef.no/projectweb/hull-to-hull/

[4] Open API Initiative (OAI): https://www.openapis.org/about

[5] Google Remote Procedure Calls (gRPC): https://grpc.io/docs/guides/

[6] Analyzing HTTPS Performance overhead: https://www.keycdn.com/blog/https-performance-overhead

[7] REST vs. gRPC: Battle of the APIs: https://code.tutsplus.com/tutorials/rest-vs-grpc-battle-of-the-apis--cms-30711

[8] Vaadin framework for web application: https://vaadin.com/