Maritime Situational Awareness Research Infrastructure (MSARI): Requirements and High Level Design

Marie-Odette St-Hilaire
Michel Mayrand
Dan Radulescu

Prepared by:

OODA Technologies Inc.
4891 Av. Grosvenor, Montreal Qc, H3W 2M2

Project Manager: Anthony Isenor
Contract Number: W7707-115137
Contract Scientific Authority: Sean Webb

DRDC-RDDC-2014-C97

Contract Report

March 2013

Abstract

The Maritime Situational Awareness Research Infrastructure (MSARI) will support the Maritime Information Support (MIS) group in its current and future research efforts related to wide area surveillance in Canada's three oceans. The major functionalities of MSARI are to acquire data from several data sources, such as AIS and LRIT data sources; store and provide a means to maintain the data; provide the capability to add applications for data processing and query capabilities to access the data. The main benefits of MSARI will be to provide easy and fast access to maritime data for the MIS scientists.

This document provides the software requirements specification that will be used as a guide for the development of the MSARI and high level design and recommendations about the software units composing MSARI. The spiral model being the software development process selected for MSARI, it is expected that the design will evolve and be refined during the implementation spiral. Moreover, some design decisions are left to the implementation phases because testing with MSA data needs to be performed.

This page intentionally left blank.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1 Introduction

The MIS group has a number of current and potential Maritime Situational Awareness (MSA) data sources:

1. Automatic Identification System (AIS) from space, land, ship, aircraft, or external systems;
2. Long Range Identification and Tracking (LRIT);
3. Radar contacts (not imagery) from space, land, ship, or aircraft;
4. Automatic Dependent Surveillance-Broadcast (ADS-B);
5. and other external data sources that the MIS group may encounter in the future that should be added to the system.

These external data sources may be accessed in a number of ways such as a streaming socket connection, periodic file upload (e.g. to/from an File Transfer Protocol (FTP) site), web service, or some other means.

The scope of this project is to design a system using the above data sources as inputs for the MIS group to support research and development in Maritime Situational Awareness. This system is referred as the MSARI.

1.1 Document Purpose and Overview

The two main objectives of this document are to

1. Provide the software requirements specification that will be used as a guide for the further development of the MSARI (sections 2 to 4) and present two use-cases to cover MSARI main functionalities (section 5).
2. Specify recommendations about the behavioural design and other recommendations affecting the selection and design of the software units composing MSARI (sections 6 to 9) and links each requirement with the software units and vice versa (section 10).

1.2 MSARI Scope

The MSARI will support current and future research related to wide area surveillance in Canada's three oceans. The design will describe an infrastructure that allows the group to input, store, and process different data sources to create outputs.

For example, if a researcher is working on AIS coverage algorithms using AIS data, the system should allow that researcher to easily access AIS data for a particular geo-spatial and temporal subset of the entire data repository, feed that data into his algorithm, and allow them to output the product of the algorithm to end users.

The main benefits of MSARI will be to provide easy and fast access to maritime data for the MIS scientists.

1.3 Intended Audience and Document Overview

This document is intended for Defence Research and Development Canada (DRDC) Atlantic scientists, namely, MSARI future users, developers and administrators (MIS group members).

1.4 Document Conventions

In this document, the following terms are to be interpreted as follows:

shall: indicates a mandatory requirement

must: indicates a mandatory requirement

should: indicates a non-mandatory requirement but justification must be provided for not meeting this requirement

will: indicates a fact or a statement of intention which is not binding

may: indicates an option

could: indicates an option.

2 Overall Description

This section provides a general description of MSARI.

2.1 MSARI Perspective

The MIS group has many heterogeneous data sources, algorithms, and visualization/output tools for MSA such as:

1. AIS, LRIT, ADS-B, radar data sources
2. Probabilistic interpolator, AIS reception mapping, AIS analysis tools, etc.
3. Google Earth, Maritime Safety and Security Information System (MSSIS) application [1], Quantum GIS, custom code, etc.
4. Data stores (MySQL, flat files on disks, Microsoft SQL server).

In that context, obtaining data for a specific geographical area, time period, and data format is time consuming and difficult at the present moment.

The main objective of MSARI is to provide easy access to MSA data for MIS team members. This infrastructure is to support the MIS research and development efforts by allowing the input, storing, and use of different data sources to create outputs.

2.2 MSARI Functionality

The major functionalities of MSARI are:

1. Collect information from several data sources
2. Store and provide a means to maintain the data.
3. Provide the capability to add applications for data processing.
4. Provide access to data.

These main functions can be viewed as four main functional components, as illustrated in Figure 1. Given this view, the requirements need to allow extension of all four components. Unanticipated data sources could be added in the future that may require new storage mechanisms; additional processing algorithms could be thought of and added to the system; and alternate output or visualisation techniques could be included.

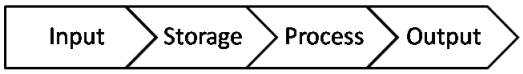

Figure 1: *MSARI main functional requirements.*

2.3 Users and Characteristics

The anticipated MSARI users are described in the following list:

1. **Internal users**: DRDC Atlantic scientists, primarily those in the MIS group who will use MSARI to perform and/or support MSA-related research and development activities.
2. **External users**: Regional Joint Operations Center (RJOC) staff who would use MSARI or tools external to MSARI to access DRDC products and data on a limited basis.

The primary users, based on the system importance for their activities and anticipated usage frequency, are MIS scientists. MSARI will be accessed by RJOC staff only in a context of DRDC products advertisement, not to provide operational data.

2.4 Design and Implementation Constraints

The items or issues that will limit the options available to the developers are:

1. Specific Application Programming Interface (API) and other interface constraints from the data sources.
2. Hardware specifications.
3. Operating System (OS) requirements such as Windows version.
4. On-site maintenance considerations.
5. Simultaneous use of the system by a minimum of six scientists.
6. Internal user access via their computers.
7. External user access to data within the infrastructure.
8. DRDC network security constraints.

2.5 User Documentation

A system guide document and the API documentation (e.g. javadoc), will be delivered along with MSARI.

2.6 Assumptions and Dependencies

These are the major assumptions (as opposed to known facts) that might significantly affect the MSARI design:

1. Open source software may be used if deemed relevant.
2. At least one year of data must be made available.
3. Total size of daily data input. For instance, it is assumed that new AIS data are available at the rate of 1 GB per day.
4. Reasonable search speed is expected.
5. To use MSARI, no knowledge of the technologies used is required.

3 Specific Requirements

This section describes the specific requirements for MSARI.

3.1 Functional Requirements

This section identifies all functionalities of MSARI.

3.1.1 Data Acquisition

These requirements address the process of collecting the information from several data sources.

a. MSARI shall provide a means to receive data from several different data sources. These data sources may include diverse AIS providers, LRIT, radar contacts from space, external data sources such as ice field databases, etc. The minimal set of the data sources to be implemented will be identified at the design review.
b. MSARI shall provide a means to parse input data from several different data sources.

3.1.2 Data Storing

These requirements address the process of storing MSA data of multiple types.

a. MSARI shall store for retrieval MSA data of multiple types. These data include ship positional and attribute data from diverse inputs (AIS and LRIT from diverse sources, radar contacts and possibly others), related ship and source metadata, and products. Here, a product is MSA data produced by a MSA algorithm. The exact nature of the data will be determined during the design review process.
b. MSARI shall provide a means to differentiate raw data from product data.
c. MSARI shall store metadata. This metadata could be related to the source of the data or the data product. Optimally, the design would follow a known standard such as ISO 19115 for geo-spatial metadata.
d. MSARI may store data in its original format as well as parsed data depending on the data type and search requirements.

3.1.3 Data Maintenance

These requirements address the process of maintaining the quality of data stored by MSARI.

a. MSARI shall provide a means to modify, retrieve and delete data in the MSARI database as well as typical maintenance tasks such as monitoring capabilities, back-ups, tune-up/optimization, integrity checks, etc.
b. MSARI shall provide a means to periodically indicate which data (if any) is older than a given time threshold. The time threshold value will be selected based on tests performed during the implementation.
c. MSARI shall provide a means to retrieve and delete data older than a given time threshold. The selection of the time threshold value will be left to the design.
d. MSARI may provide a *state board* showing which feeds are up or down, data throughput, trends over time, users connected to MSARI, etc. The exact information to be displayed will be determined in the design review process.

3.1.4 Data Processing

This requirement addresses the capability to add data processing applications to MSARI.

a. MSARI shall provide the capability to add data processing algorithms. The algorithms products may or may not be stored by MSARI. This implementation choice depends on the pertinence of the algorithm products for the users.

3.1.5 Data Access

These requirements address the process of accessing MSA data.

a. MSARI shall provide the user with a means to access data based on location, time, and parameters specific to the data set. These parameters shall include:
    i. Ship attributes (e.g. name, MSSI, flag...).
    ii. Time attributes (at least time intervals).
    iii. Position (at least bounding box coordinates).
    iv. Source attributes (at least name and data type).

    The parameters choice will be left to the design.
b. MSARI shall provide the user with a means to get data with different delivery methods such as daily extraction, ongoing streaming data, downloading, etc. The delivery methods will be tested and specified during the implementation process.
c. MSARI shall provide a means to inform the user that data access requires more time than a given time threshold. The time threshold value will be selected based on tests performed during the implementation.

d. MSARI shall provide the user with a means to access the data processing algorithms products. Products may or may not be stored by MSARI. In the latter case, an access to these products may be made available to the users by MSARI or may simply be accessed separately.

e. MSARI shall allow the internal user to access data from their computer. See section 2.3 for internal user definition.

3.1.6 Extensibility

There is a requirement for extensibility of MSARI, i.e. to minimise the level of effort required to implement extensions to MSARI.

a. MSARI shall provide the capability to add new data sources (e.g. the same type of data but from a different provider).

b. MSARI shall provide the capability to add new and extend existing data types for storing.

c. MSARI shall provide the capability to add processing algorithms.

d. MSARI shall provide the capability to add new means to access data.

3.2 User Interfaces Requirements

MSARI shall provide a means to access and maintain the data. The following sections describe the requirements for the user interface allowing data access and maintenance.

3.2.1 Access

MSARI shall provide the user with a graphical means for accessing data. The following describes the requirements for the user interface exposing the data access functionalities described in 3.1.5.a to 3.1.5.d.

a. The user interface shall provide a means allowing querying on at least 4 categories of attributes (ship, time, position and source) and a means (e.g. a button) to allow the user to send a request to get data based on the selected parameters.

b. For each query category, MSARI shall provide a means to configure the parameters describing the data required.

   The following provides a set of possible search parameters for each query category. Note that this list is not complete and other parameters may be considered while others discredited, depending on the data sources integrated.

i. Ship: Identification Number (ID)s such as Maritime Mobile Service Identity (MMSI), International Maritime Organization (IMO) and callsign, flag, and ship name. It may also include ship dimensions and types.

ii. Time: time intervals and time periods (e.g. first week of each month for a given year)

iii. Position: bounding box (IN or OUT) and boundaries.

iv. Source: source name and data type.

These parameter descriptions should allow the use of wildcards and may be related by a choice of AND and OR logical operators. Each of them should also be additive, e.g. instead of just one ship ID, allow search on multiple ship IDs or a range of IDs.

c. MSARI shall provide the user with a visual indicator, such as a progress bar or a wait cursor, if data access requires more time than a given time threshold.

d. In the case processing algorithm products are stored by MSARI, then MSARI shall provide the capability to add a means to allow the selection of the MSARI processing algorithm.

e. The user interface could contain:
   - A map based query interface to define bounding boxes.
   - An interface allowing
     – the selection of raw versus parsed data
     – the selection of the output data format (e.g. Comma-Separated Values (CSV) for parsed data and text for raw data).

3.2.2 Maintenance

MSARI shall provide the user with a *state board*, i.e. a graphical means to provide information about the state of MSARI. The following describes the requirements for the user interface exposing the maintenance functionalities described in 3.1.3.d.

a. The user interface shall display which data sources are up or down, data throughput, trends over time, users connected to MSARI, etc. The exact information to be displayed will be determined in the design review process.

b. The access to the maintenance user interface may be restricted to a dedicated administrative user only. The access may be secured with a password unknown by the other users.

3.3 Software Interface Requirements

MSARI must interface with five types of software: data sources, data stores, open source software, data processing algorithms and operating system. For each type, the exact software to interface with is still to be determined during the design review process. Some examples for each type are given in the following enumeration.

1. Data sources:
    a. AIS data sources (e.g. MSSIS [1]),
    b. LRIT data sources,
    c. Ice and other geo-type data sources.
2. Data stores: Database (DB) or other storing capabilities for data archiving:
    a. Database (including possibly Geo-DB),
    b. Flat files.
3. Other open source software:
    a. Java libraries,
    b. Web services.
4. Data processing algorithms:
    a. Visualisation applications,
    b. Probabilistic interpolator,
    c. AIS reception mapping.
5. Operating system:
    a. Windows

Table 1 describes the purpose and services needed for each interface. The interfaces between all functionalities of MSARI shall be made through APIs.

All implementation choices about software interfaces are left to the design review process.

Types of Software to Interface With	Interface Purpose
Data sources	Connect and receive the input data (raw data) from the source.
Data stores	Data management: store, get and delete data.
Data processing algorithms	Connect the algorithm to MSARI: get results for a given set of data.
Open source software	Connect the software to MSARI.
Operating system	Provides computer hardware management and common services for MSARI applications.

Table 1: *Software Interfaces General Description*

4 Non-Functional Requirements

This section describes the non-functional requirements for MSARI.

4.1 Performance Requirements

These requirements address MSARI's performance.

a. MSARI shall provide reasonable response time, i.e. process all queries at a reasonable speed, in standard conditions. If response time is higher than a given time threshold, there is requirement to inform the user about it (see 3.1.5.c).
b. MSARI shall support a reasonable number of users (and their threads) working simultaneously, in standard conditions. More precisely, it shall support minimally the MIS group, which totals 6 users.

4.2 Software Quality Attributes

This section specifies additional quality characteristics for MSARI that will be important to either the users or the developers and administrators.

4.2.1 Usability

MSARI data access functionalities shall be easy to use. To access MSARI data, no knowledge of the technologies used in MSARI should be required. For instance, the user should not need to be proficient in Structured Query Language (SQL) language to retrieve data satisfying a given set of geo-spatial and temporal parameters. On the other hand, this feature should not restrict SQL knowledgeable users from using the full querying power of SQL.

4.2.2 Reliability

MSARI shall be available and responsive to the required activity. This means that MSARI should provide repeated and consistent data delivery, acquisition and storing.

4.2.3 Extensibility

The extensibility requirement is captured by the functional requirements given in 3.1.6.

4.2.4 Integrity

MSARI shall provide a means to ensure that data stored is accurate, valid, and consistent. Data maintenance functional requirements, enumerated in section 3.1.3, cover part of this quality requirement. Additionally, in order to preserve data integrity, MSARI data model should be designed following data modeling standards. Data modeling and standards choices will be determined when the minimal set of data sources will be defined.

5 Use Cases

This section provides two use cases which cover some of the main MSARI functionalities described in section 3.1. A use case defines a goal-oriented set of interactions between external actors and MSARI.

The first use case covers MSARI data acquisition, storing and access. The second covers the data acquisition, storing, processing and access.

5.1 Use Case 1: Data Acquisition, Storing and Access

An MIS group member tries to find patterns in AIS and LRIT data corresponding to a given geographical area and a given period of time. This person is developing an algorithm to process AIS and LRIT data corresponding to the criteria.

There are 2 possibilities:

a. The user accesses the MSARI data query form and enters the information required for the query: type of data (AIS and LRIT), data sources names (optional), time period and geographical area coordinates. The user sends the query and waits for a response, which is downloaded onto his local hard drive as a CSV file, and uses the output to feed the algorithms.
b. The user uses the MSARI API to interface with the system within the algorithm. The query parameters are set programmatically and communicated to the system as parameters. The user runs the algorithm which uses the data.

Figure 2 shows the interaction between the main MSARI classes of functionalities. It describes the data flow (illustrated by arrows) between these functionality classes for this use case. Note that it does not describe a software architecture.

The MSARI functionalities are encompassed by the dashed line. The data sources box, which is not part of MSARI, represents the diverse data sources MSARI has to interface with.

The Acquisition, Storing and maintenance, Access and Processing boxes correspond to the following functionality requirements respectively: data acquisition (3.1.1), data storing (3.1.2) and maintenance (3.1.3), data access (3.1.5) and data processing (3.1.4).

For this use case, AIS and LRIT data are acquired from diverse data sources, parsed and stored by MSARI. When the user requests it, the AIS and LRIT data, corresponding to the input geo-spatial and temporal user-defined parameters, is retrieved. It is made available to the user either by the User Interface (UI) (as described in a) or directly by the API (as described in b).

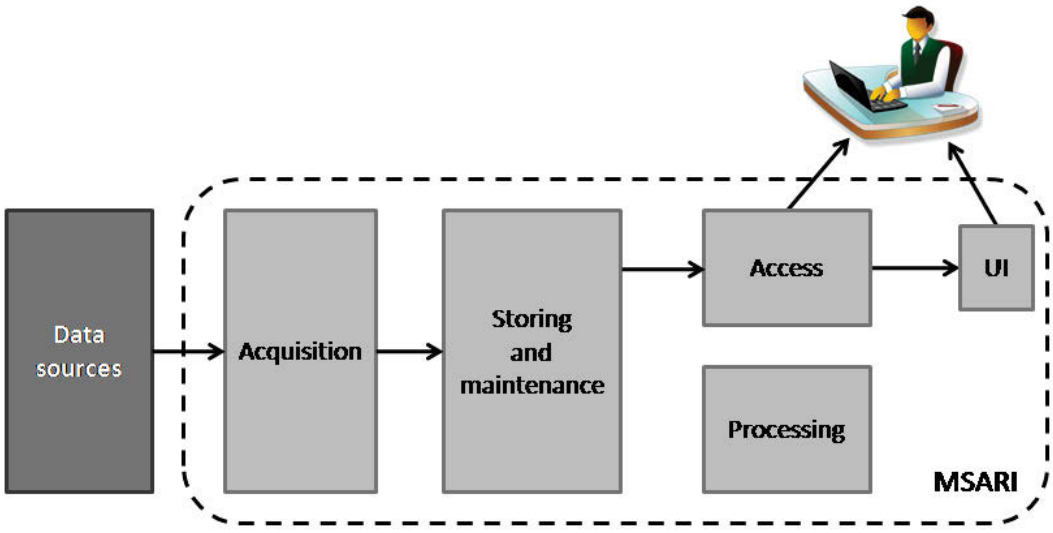

Figure 2: *Data flow between main functionality requirements for use case 1.*

5.2 Use Case 2: Data Acquisition, Storing, Processing and Access

An MIS group member has an algorithm that requires a constant feed of data from a specific area of the world. Every 4 hours, the algorithm will produce an output.

The algorithm products may or may not be stored by MSARI, and it may or may not be available to other MIS researchers. The following cases cover these options:

1. products not to be shared;
2. products to be shared but not stored;
3. products to be shared and stored.

In the first case, the scientist may use MSARI only to test his algorithm or produce output for his research. In the second case, the scientist may want some other MIS researchers to access the products, without storing it by MSARI. The last case would occur if products are deemed relevant for the entire MIS group.

Figure 3 shows the interaction between the main MSARI classes of functionalities, for case 3 of this use case. Note again that it is not a software architecture. The black arrows describe the data flow from the data sources to the algorithm. The dashed blue arrows represent the data flow in the third case where the algorithm output (part of the data processing component) is stored by MSARI.

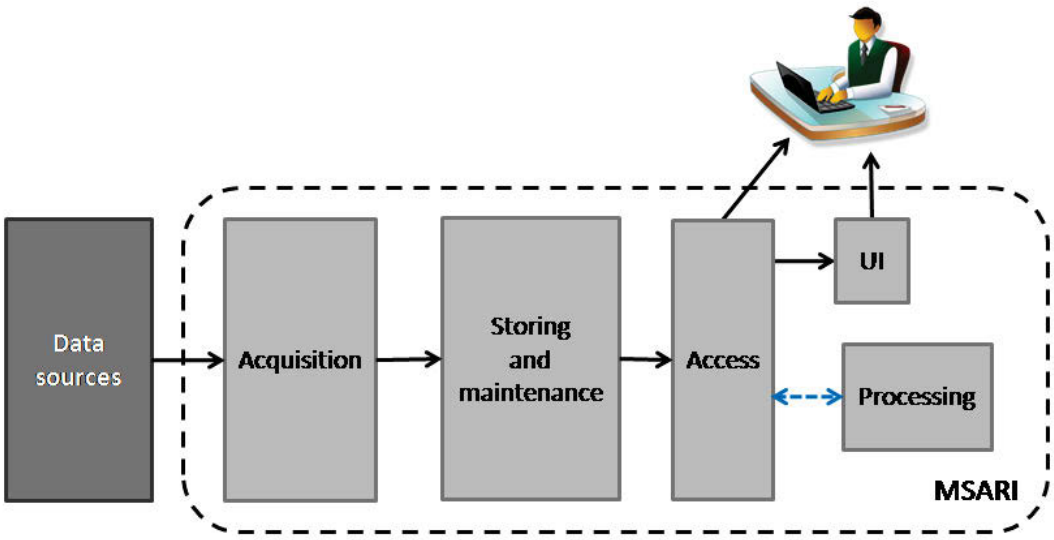

Figure 3: *Data flow between main functionality requirements for use case 2.*

6 Global Design Architecture

This section presents the MSARI-global design architecture, that is, decisions about MSARI's behavioural design (how it will behave, from a user's point of view, in meeting its requirements, ignoring internal implementation) and other decisions affecting the selection and design of the software units that make up MSARI.

From a user point of view, MSARI allows users to obtain data, based on temporal and spatial criteria, from diverse sources via their desktop. In other words, MSARI acts as an interface to MSA data acquired from several data sources, such as AIS from diverse providers, LRIT, etc.

6.1 Functional Components

In order to provide the user with access to quality data, processed or not, MSARI needs capabilities to acquire, store, maintain, process and deliver data. As a consequence, MSARI design is divided into five main components representing each of these capabilities. Figure 4 illustrates these components in interaction with the external elements: users and data sources. The arrows indicate the data flow.

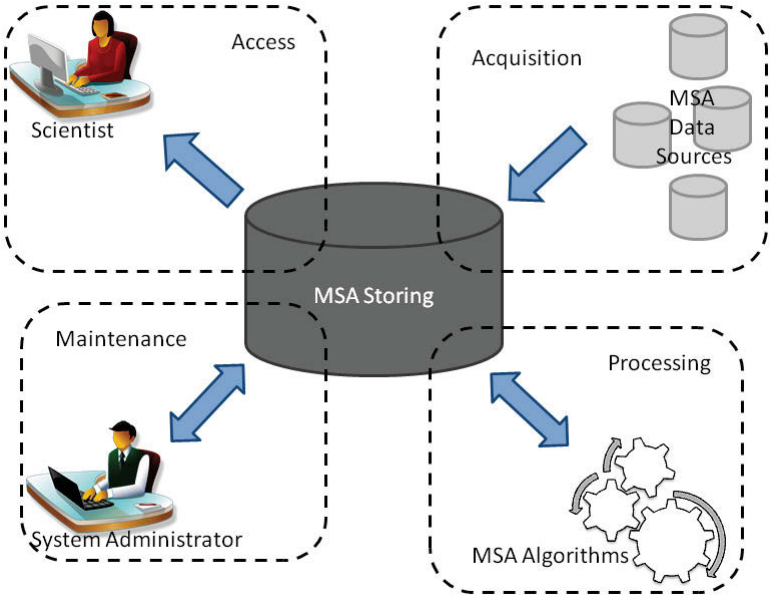

Figure 4: *Functional view of MSARI: data acquisition, data storing and management, data processing and access.*

However, MSARI should not be viewed only as a data repository: it is an infrastructure, meaning that it will evolve. Concretely, it implies that each component may change in the future to satisfy DRDC Atlantic research activities. These modifications may include:

1. Acquisition: more data feeds.
2. Storing and maintenance: more data types.
3. Processing: more processing algorithms.
4. Access: more sophisticated query capabilities.

To meet this extensibility requirement, components are designed to be independent and only rely on the storing component. For instance, if some functionalities of the data acquisition component are down, access to data is not perturbed. And conversely, if data access is somehow impossible, it does not disrupt the data acquisition or the data processing. This loosely coupled design also improves reliability as described in 4.2.2.

6.2 Open Source Software

MSARI design is oriented towards the use of robust open source software. This strategy aims to benefit from tools already developed, tested, maintained and approved by the community, which in return lowers the risks associated to MSARI development and maintenance. It also allows focusing on the data storing and acquisition components, which are the basis of MSARI.

Figure 5 illustrates how this design choice impacts MSARI development. The open source strategy is directly applied on the maintenance and access components (see sections 7.3 and 7.4) where database front-end tools are proposed. Because the processing component is a capability to add processing algorithms to MSARI, it reduces the implementation efforts to interfacing between the data storing and algorithms (see section 7.5 for details). That way, the focus for the development is on the data acquisition and storage components.

6.3 Data Exchange

Data exchange is an important part of MSARI: from sources to storage, from storage to users. Several strategies exist for heterogeneous data exchange in remote environments. Because MSARI involves databases, the most common choices are: direct data access with Database Management System (DBMS), web services, FTP and routing engines such as Apache Camel [2].

For the stored data access, the simplest and most powerful way is to directly rely on the DBMS. It allows remote and controlled access with the power of the database query language. It is also the fastest way to access data because it does not involve other processes.

The second option is web service. By exposing a database as a web service, it allows data to be consumed easily by other applications. For instance, if it is decided that a custom query interface is required for MSARI, web services would be the best choice (rather than FTP

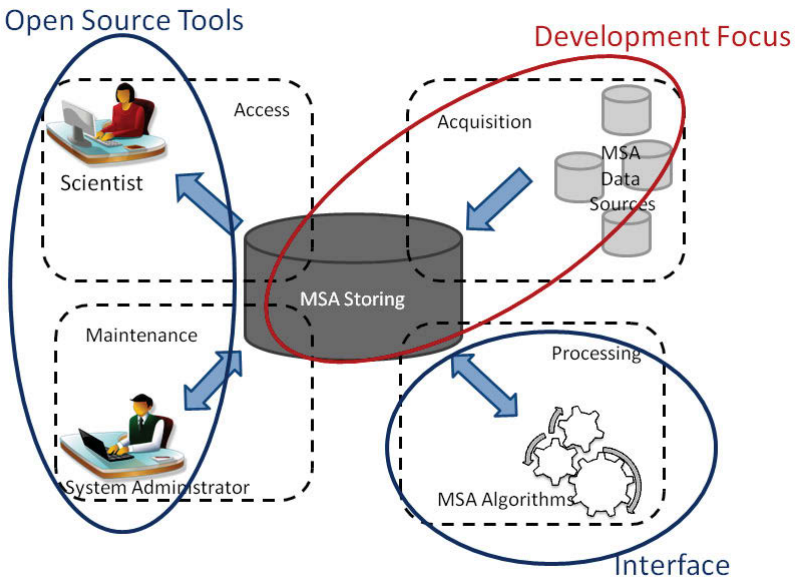

Figure 5: *Functional view of MSARI: development strategy.*

for instance). Several reasons, such as interoperability, open standards and protocols, open source efficient related technologies and large developer communities, justify this choice. However, web services should be developed only when required. It is well known that web service interfaces require more development efforts than non-web interfaces.

Because of the requirement of filtering the data based on space and time parameters and the need to align the data, the current FTP file retrieval technique is not sufficient for supporting the preprocessing needed.

An engine like Apache Camel is also an interesting choice because it allows configuring the routing of different data inputs and outputs, such as databases, CSV files, web services, FTP, etc. In the particular case of MSARI, since there is a storing requirement, all routes would pass by the databases (origin or destination). In that sense, the use of Camel would not be optimal. Moreover, Camel would add an extra layer to the database access, risking limiting or slowing the data access.

6.4 Programming Language

MSARI is intended to evolve, so it should be accessible to a wide number of computer scientists and students. Because Java is the most widely used language in enterprise, it is thus the most appropriate programming language for implementing the software units of MSARI. Moreover, it provides numerous open source libraries for supporting databases, web services and many different computing categories. However, the use of Java as the

implementation language for MSARI does not limit the development of MSA algorithms to Java. Algorithms developed in other languages can be integrated easily with a Java wrapper.

The object-relational mapping Hibernate [3] (and its spatial extension Hibernate Spatial [4]) will be used instead of Java Database Connectivity (JDBC) to connect, query and update the database. Hibernate is an Object-Relational Mapping (ORM) solution for Java. It is a high performance object-relational persistence and query service which allows developing persistent classes following object-oriented idiom. The decision to use Hibernate instead of JDBC was made in the perspective that MSARI will evolve. The use of Hibernate speeds up the development and cuts down maintenance cost by reducing the required coding to interact with the DBMS. Moreover, Hibernate eases connection management, connection pooling and caching.

7 Architectural Design

This section identifies the software units that make up MSARI. Each software unit regroups functionalities from the same process and may be mapped to one or more Java classes. Each software unit is linked to requirements described in sections 3 and 4.

The MSARI architecture is divided into five main functional components: data acquisition, storing, maintenance, access and processing. The software units are grouped by functional components as illustrated in Figure 6.

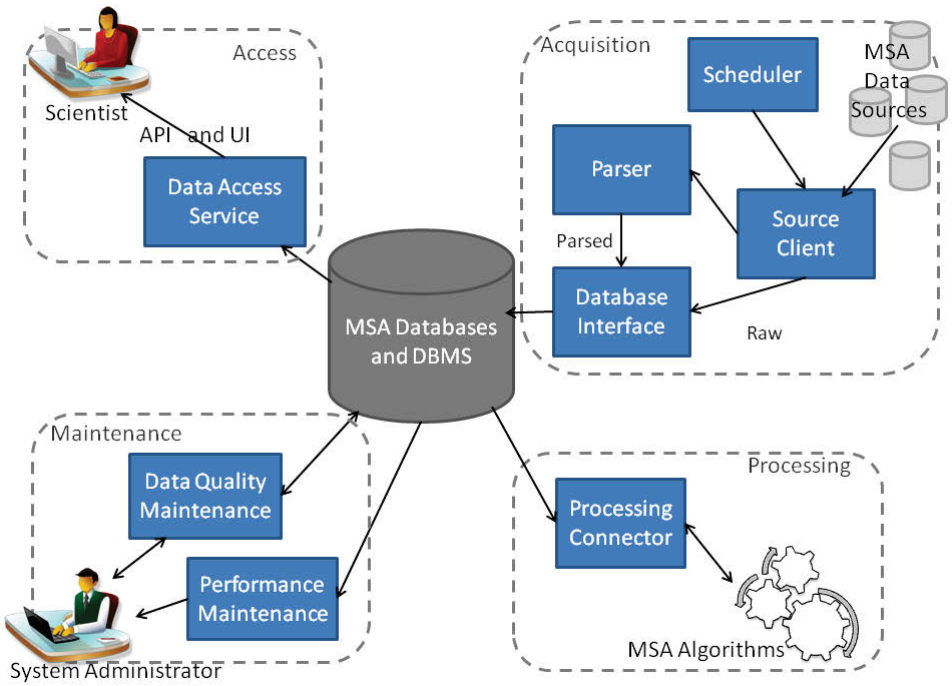

Figure 6: *MSARI's functional software units. The blue boxes represent the units while the arrows are for the data flow.*

7.1 Data Acquisition

The data acquisition component is responsible for the collection of raw data from all sources, and the generation of a common data format that is sent to the database for persistent storage.

The data acquisition component abstracts the specifics of each data source, meaning that it is adaptable to various input types, be it a website, live streaming data, a local text file, etc. The design encapsulates the specifics of each raw source into low-level components, thus minimizing the impact of adding new sources as well as the amount of code required

for the task. Figure 7 shows the data acquisition pipeline detailing the flow of data from raw sources to the local database storage. The following sections detail the critical subcomponents involved in data acquisition.

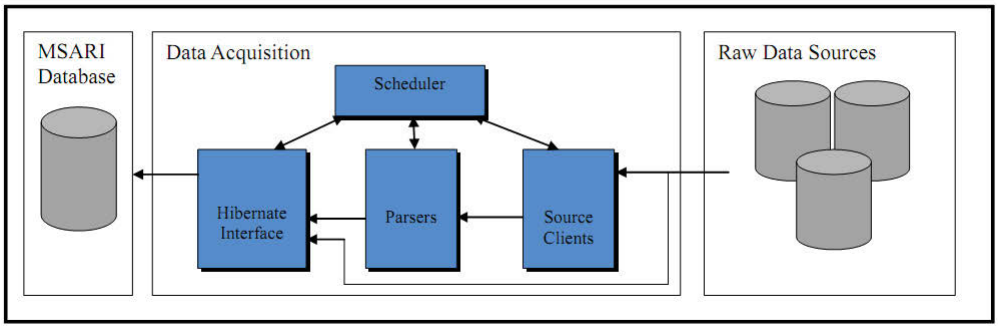

Figure 7: *The diagram above shows the main subcomponents involved in acquisition.*

This design satisfies the extensibility requirement 3.1.6.a.

7.1.1 Scheduler

The scheduler controls all subcomponents of data acquisition. As such, it is an observer to all components under its control, receiving notifications on their status. The scheduler controls when to start or stop a component, the number of a component's instances required and the notification structure. It is also through the scheduler's public methods that the user should control the acquisition process.

The scheduler is linked to requirement 3.1.1.a.

7.1.2 Source Client

The source client encapsulates all communication with raw data sources. It receives data and forwards it out in a format that the parser can interpret. In the context of the observer design pattern [5], the source client takes the role of a subject. Being threaded, the design allows several source clients to output to the same parser.

Consider a source client receiving data from a remote AIS source through a UNIX socket. At any time, a second source client reading AIS data from a local file can be connected to the same parser without altering any of the parser's code. The only requirement is for the parser to subscribe to the new source client. This configuration is shown in Figure 8.

The source client is linked to requirement 3.1.1.a.

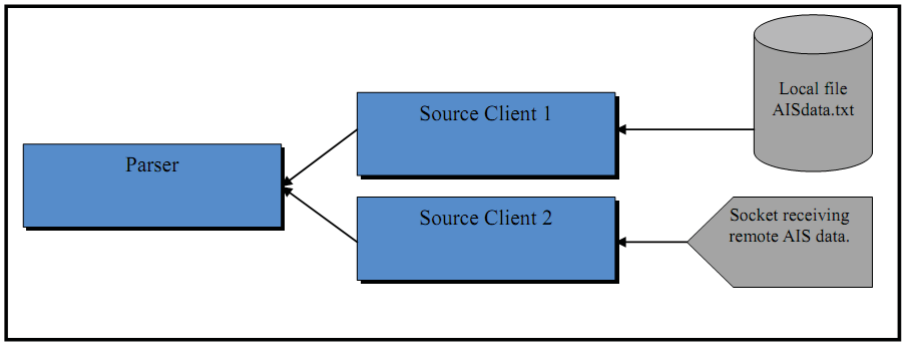

Figure 8: *The diagram above shows the flexibility of the interaction between Parser and Source Client. Here, two clients with completely different sources output AIS messages to a single parser. There is no theoretical limit to how many clients can output to the same parser but in practice, latency limitations of the parser must be taken into account so as not to bottleneck the pipeline.*

7.1.3 Parser

The parser receives data from the source client, decodes it and creates an object that is aligned with the MSARI database model. The parser is built around the strategy pattern [6] (see Figure 9), it contains a single decoder and a corresponding mapper, both encapsulated in separate implementations. The parser is adaptable because decoders and mappers can be changed dynamically at runtime and is extensible as any new decoder or mapper requires only writing a new decoder or mapper class without modifying any of the higher lever components. New instances of parsers can be added as needed. Within the observer pattern paradigm, the parser acts as observer to a source client.

In order to maintain the simplicity of the source client code, it is recommended that a single parser observe a single source client. It is possible however for several parsers to observe a single source client. For example, when reading AIS messages from a local file, the parser becomes a performance bottleneck because it needs to wait on Hibernate to complete its transaction with the database. In that case two or more readers can be assigned to the same source client. The source client can then cycle through all available parsers and notify each one in turn with a different message as shown in Figure 10.

The parser is also responsible to filter bogus and non available data. In order to preserve the integrity of data provided by MSARI, parsed data containing error or non available data is not stored. For example, AIS message with a longitude value of 181 degrees indicates that longitude is not available and is the default. In that case, in order to save disk space and thus avoid slowing data access (see section 8.3 for details about query optimization) the longitude is not stored. Other examples include empty strings or dates that are out of range such as a month value of 15. However, this bogus data is available in the raw message,

which is always stored. At the moment, once the parsed and raw data are stored, there is no means (e.g. - a flag) to indicate if a raw message contains bogus data.

The parser is linked to requirement 3.1.1.b.

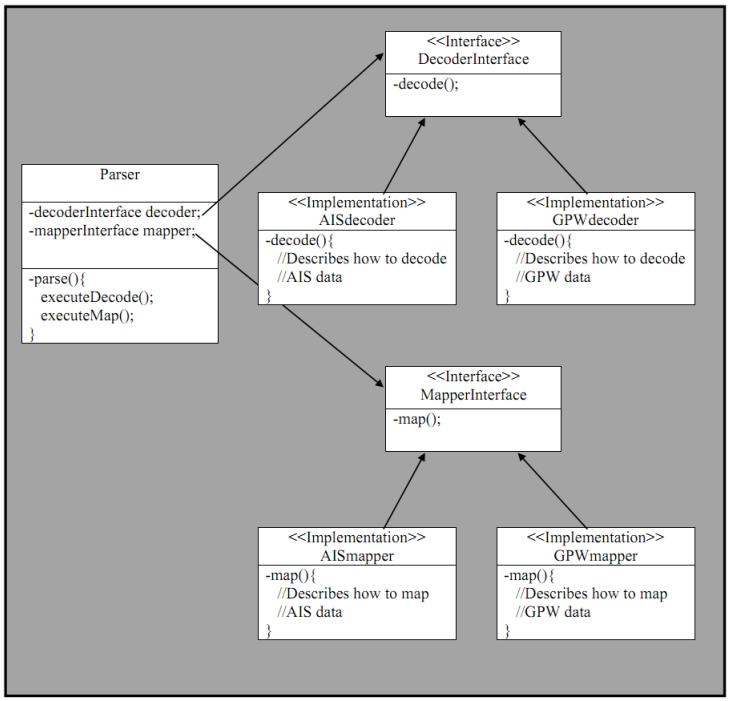

Figure 9: *The above diagram describes the strategy design pattern as used by the parser. When creating a parser, the decoder and mapper variables are instantiated polymorphically using specific implementations such as the AISdecoder class or PGWdecoder class. The parser only executes the decode and map commands without caring about their assigned implementations. These variables can also be set dynamically. This design maximizes the flexibility of parsers by removing them from the specifics of the raw data.*

7.1.4 Database Interface

The database interface updates the MSARI databases with new data. It stores the data source information (parsed and raw) and MSA data processing algorithm products (if any). It is also responsible to update the corresponding metadata.

The implementation of this component is done with Hibernate. The Hibernate library and drivers offer transparent methods for communicating with a database. Hibernate's most important advantage however is that it abstracts the layer in charge of translating a Java object to a database. A single mapping file is required to establish the relationship between a Java class requiring persistence and its corresponding database model. Hibernate then takes care of generating all required SQL commands to store the object in the database.

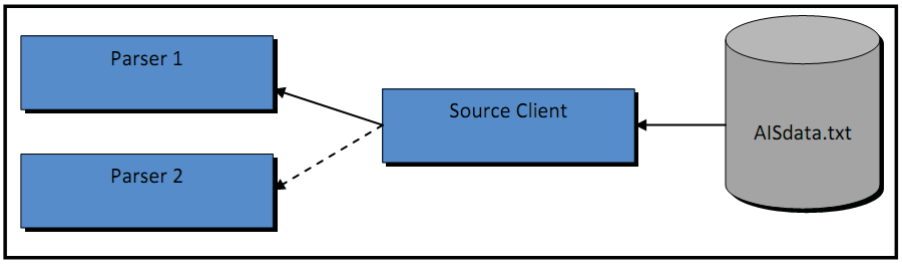

Figure 10: *The diagram above shows two Parsers receiving data from a single client. While this is possible, the client must be tailored to output a different message to each parser. Within the current design this must be done programmatically for each specific client. It is therefore suggested that this configuration be avoided and that the number of parser subscriptions to a source client be limited to one.*

This abstraction removes the need for a great deal of code but has the drawback of adding latency to the acquisition chain.

The database interface component is linked to requirements 3.1.2.a, 3.1.2.c and 3.1.2.d.

7.2 Data Storing

The data storing functional component stores for retrieval MSA data of multiple types.

The data storing functional component is a data store containing and structuring the following items and the relations between them:

1. Metadata;
2. Raw data;
3. Parsed data.

The metadata is used to describe the data sources and processing algorithms products. The source metadata includes, but not limited to, the data source name, type of data, date of creation and update of the data set. The latest update time can be used to assess the source's update rate. For another example of use of the source metadata, consider the case of Ottawa space-based AIS. Integrated as a source, the Ottawa AIS space-based reports would be stored by MSARI and for each of them there would be a link to the information about the source (could be detailed in other tables). The products metadata, which includes at least the data source name, the product name, type of data, author and the timestamps of the last processing, is used, amongst others, to differentiate the processed from the unprocessed information (in order to avoid, for instance, data incest). In both cases, metadata is used as a means to assess and maintain data quality. Also, metadata can be broaden to include either additional standard field or other user-defined useful information.

The raw data is the data original format (as provided by the source). Every data input is stored and accessible for retrieval.

All the information contained in the raw data is parsed in order to be available for query. The relations between the parsed, the raw data and the source are preserved.

Because data is stored for retrieval, databases and DBMS have to be involved.

Design decisions for the storing component, such as the number of databases and the choice of DBMS, strongly depends on the characteristics of the data to be stored. Based on these characteristics, section 7.2.1 proposes a DBMS.

The logical and physical data model are described in section 8.

7.2.1 DBMS

The most popular open source relational DBMS are PostgreSQL [7] and MySQL [8]. PostgreSQL is an object-relational database management system well suited and efficient with very large and complex databases. It has a spatial extension called PostGIS [9] which adds support for geographic objects and based on many standards (one being OpenGIS). PostGIS makes PostgreSQL the best SQL candidate for MSARI, because it is more mature than the MySQL spatial extension [10].

On the NoSQL side, several options are also available. For instance, MongoDB [11] has gained popularity as a high-efficiency DBMS. It has the ability to scale horizontally efficiently and to support very large databases. MongoDB is particularly well suited for cloud implementation (but cloud computing is not an option for MSARI). It supports geo-spatial indexing [1] but it is not yet as mature as PostGIS. For instance, it is limited to 2 dimensional data.

PostgreSQL with its PostGIS extension is the selected DBMS for MSARI. To ease maintenance and access, if more than one database are used, PostgreSQL will be selected for all databases.

The DBMS component is linked to requirements 3.2.2.b, 4.1.a and 4.1.b.

Database Access Restrictions

PostgreSQL manages database access permissions using the concept of roles. A role can be thought of as either a database user, or a group of database users, depending on how the role is set up. Roles can own database objects (tables for instance) and can assign privileges on those objects to other roles to control who has access to which objects. Furthermore,

1. http://www.mongodb.org/display/DOCS/Geospatial+Indexing

it is possible to grant membership in a role to another role, thus allowing the member role use of privileges assigned to the role it is a member of.

For MSARI, non-administrators have the same role for read-only access to data. Administrator role will be created for read-write access for maintenance purpose and for the database manager. This feature is linked to requirement 3.2.2.b.

7.3 Data Maintenance and Monitoring

The data maintenance functional component is responsible of maintaining the quality of data stored by MSARI. The goal of maintenance is to assess and preserve:

1. Reliability of MSARI, as described by requirement 4.2.2, or more generally MSARI's performance.
2. Quality of stored data.

7.3.1 Performance Monitoring

The performance monitoring unit analyzes the state of MSARI: if sources are alive, statistics about them, users connected, etc. However, the main focus of this unit is the acquisition performance and data throughput which is assessed based on the source metadata (e.g. data inputs timestamps and size). See section 9.2 for a description of the user interface to the performance monitoring unit.

The performance maintenance is linked to requirement 3.1.3.d.

7.3.2 Data Quality Maintenance

The data maintenance unit:

1. Allows one to modify, retrieve and delete data.
2. Periodically indicates which data (if any) is older than a given time threshold.

Data Quality Maintenance Implementation

This software unit is not mapped to java classes. Instead, an open-source software is used to cover the functionalities. As PostgreSQL is the MSARI's DBMS, pgAdmin III [12] is the appropriate choice for data quality maintenance.

Several reasons justify this decision in the context of MSARI (see [12] for a detailed description):

- Most popular open source management tool for PostgreSQL.
- Supported on MS Windows, GNU/Linux, Mac OS X, Solaris and others.

- Features fast, multi threaded query and data editing tools and support for all PostgreSQL object types.
- Includes
  - a graphical administration interface,
  - an SQL query tool and a graphical query builder,
  - a procedural code editor.
- Supports all PostgreSQL features.

Dumps and exports, based on criteria such as data timestamps older than a given threshold, can be performed with this tool. Also, maintenance activities reducing the size of the database, such as vacuuming, re-indexing and analyze, can also be performed with it. Since it is assumed that data quality maintenance is performed by a SQL literate person, all data quality maintenance activities required can be performed with pgAdmin III.

Another useful administrative open-source tool, pgsnap [13], could be used in combination with pgAdmin III. It is a PostgreSQL report tool, which creates static HyperText Markup Language (HTML) performance reports for a database.

The maintenance process, including dumps for data older than a time threshold, vacuum, re-indexing and analyze, is documented in the MSARI system guide [14].

The data quality maintenance is linked to requirements 3.1.3.a, 3.1.3.b and 3.1.3.c.

7.4 Data Access

The data access functional component is responsible of querying and delivering MSA data. It allows the user to get MSA data corresponding to various criteria, including geo-spatial attributes. Access to data can be made either programmatically (from a software component external to MSARI, but part of the DRDC network) or with a user interface.

7.4.1 Data Access Service

The data access service is the software unit allowing access to MSARI data.

The design of the data access service is based on the assumption that:

1. users interested in accessing MSARI data from an API are SQL literate
2. users mostly interested in accessing MSARI data from a UI may not be SQL literate

Based on that assumption, the decision to develop or not an SQL abstraction layer depends on the quality of the open source graphical query builder tools for PostgreSQL and PostGIS.

Since the MSARI design is oriented towards the use of open source software, the approach is to test and compare the existing tools with MSARI data store and data. If no query

builder tools (or combination of) for PostgreSQL and PostGIS is found satisfactory for the MIS group members needs, the layer will be developed. In that case, the data access service will be a web service exposing some of the SQL abstraction interface.

But since the addition of such a layer could eventually limit the query options and force a re-design of the GUI if new data sources are added (which limits MSARI extensibility), the *no layer* option is favoured. In that case, the data access service is simply the DBMS engine. See section 9.1 for details about the data access user interface.

The data access service is linked to requirements 3.1.5.a, 3.1.5.b, 3.1.5.c, 3.1.5.d and 3.1.5.e.

7.5 Data Processing

The data processing functional component allows adding data processing algorithms to MSARI. Algorithm products may or may not be stored by MSARI, and may or not be accessible to other users.

As mentioned in section 5.2, three cases involving data processing are identified for MSARI:

1. products are not shared;
2. products are shared but not stored;
3. products are stored and thus shared.

In the first case, the user needs to use MSARI data to feed its standalone processing algorithm. MSARI data is accessed using the data access service (simply using JDBC driver). For this case, the processing is outside MSARI. No implementation of the data processing functional component is required. It is the responsibility of the user to use the data access service accordingly.

The second case is similar to the first one, but the algorithm products have to be available to other users without being stored by MSARI. Amongst the possible strategies, the user could decide to expose the interface of its algorithm as a web service. That way, the products would be available to the MIS group members, via their computers (as long as it is part of the DRDC Network). The user could also save the algorithm results in another database (not MSARI). That way, results would also be available to other scientists via a client using JDBC for instance. This assumes that the database can be accessed from another machine [2]. Again, for this case, the processing is outside MSARI and no implementation of the data processing functional component is required. It is the responsibility of the user to use the data access service accordingly and to broadcast its products.

The third case, the processing algorithm would have been deemed reliable and pertinent by the MIS group. Products of this algorithm would be stored and updated by MSARI

2. Note that accessing remotely a Microsoft Access database is very tricky.

and therefore accessible for queries. To achieve data processing in that case, a connection between the MSARI data storing component and the algorithm is required. This interface is provided by a processing connector.

7.5.1 Processing Connector

The processing connector interfaces between any MSA algorithms and the MSARI storing component. It provides the MSA data required by the algorithm and stores the algorithm products along with the required product metadata. Each processing algorithm added to MSARI will implement that interface.

The processing connector component is linked to requirement 3.1.4.a.

8 Data Model

The MSARI data store contains positional information with timestamps (vessels and airplanes reports), potentially geometric objects (e.g. maps, boundaries, track models, etc.) and other algorithm products such as coverage estimate matrices. All positional data have a source name, a timestamp, a position, an ID and different sets of attributes, depending on the source (vessel name, flag, vessel type, etc.).

One of the most important characteristics of the data is its rather large throughput (approximately 1 GB per day). This characteristic pushes the maintenance design and the performance optimization to the very first position in the priority list. But the main advantage of this data is the low relational content and their independence (setting aside time and position).

Section 8.1 presents the data modeling approach, section 8.2 describes the conceptual and logical data model while section 8.3 describes the physical data model, i.e. the implementation of the logical data model for the DBMS.

The physical and logical data models are linked to requirements 3.1.2.a, 3.1.2.b, 3.1.2.c, 3.1.2.d and 4.1.a.

8.1 Modeling Approach

It is clear that a poor logical data model would limit MSARI's extensibility, reliability and integrity of its data. The key aspects for this design are:

1. Maintainability: simple structure for a large daily throughput;
2. Extensibility: capability to add new data sources, data types and processing algorithms products with minimal impact;
3. Scalability: structure allowing the content to grow to very large size while supporting the transactions in reasonable delay.

The approach used for data modeling was to first investigate if and how existing data models could be reused for MSARI. Amongst the possible data models for the maritime domain, were the data models of:

- Service Oriented Development Architecture (SODA) which is the evolution of the MUSIC architecture [15],
- National Information Exchange Model (NIEM)-Maritime [16],
- Rapid Environmental Assessment (REA) database [17],
- 2009 United States AIS Database [3],
- PASTA-MARE project database [18] and [19].

3. http://www.marinecadastre.gov/AIS/default.aspx

Some characteristics of the SODA data model make it the most interesting for MSARI:

- Designed for the context of maritime data sources integration;
- Simple and extensible (addition of new data sources/types - e.g. HFSWR - is rather simple);
- Data has low relational content as for MSARI.

However, three important aspects make it impossible to use it *as is* for MSARI.

- Designed for track management - MSARI data is report centric;
- Not designed for user access: database is accessed only by applications.
- May not be optimized for large throughput.

The rather flat structure of NIEM-Maritime [20] makes it less interesting. However, the use of NIEM-Maritime vocabulary can be pertinent for MSARI because it is recognized through the MSA community. Some of the NIEM-Maritime vocabulary was found to be very verbose for this application, e.g. *VesselIMONumberText* for the IMO number.

Even though the REA database is mostly for environmental data, the naming convention (upper/lower cases, plural, etc.) is very consistent and is pertinent for MSARI.

The 2009 United States AIS and PASTA-MARE project databases, exclusively designed for AIS, were found too restrictive for MSARI where other types of data are stored. However, some lessons learned of the PASTA-MARE project were found pertinent for the physical data model design (see section 8.3.2).

8.2 Conceptual and Logical Data Model

Figure 11 displays the conceptual model. In this graph, rectangles represent the entities and the ovals are the properties of the entities. The arrows designate the relation between two entities, e.g. report has a data type.

This model has two main properties:

1. report-centric,
2. attributes are modeled using the Entity-Attribute-Value (EAV) model [21].

The `report` entity is the center of this partial star schema [22]. A report is a broadcast element and the entity type describes its nature: vessel, aircraft, base station, aid-to-navigation or other. Each report is acquired by MSARI as a raw message, at a given time (`MSARITimeStamp`) from a given source and in a given data type. If this raw message was successfully parsed by the MSARI parser, the `isParsed` flag is set to true and its position, time stamp (`ReportTimeStamp`) as well as the other attributes describing the report, like MMSI, course, message type and many others, are persisted. The simplicity of this model allows adding other satellite information related to a report without impacting the structure. For instance, adding algorithm products to that model will not impact

the current structure because products are usually based on report data (not the inverse). For example, a tracker uses reports to produce tracks and these tracks can be represented within the MSARI database as `tracks` entities without having to add attributes to any of the report-related entities.

The EAV model, also called the vertical model, was selected to model the report attributes (other than position and timestamp) because the number of attributes that can be used to describe the reports is potentially vast, but the number that will actually apply to a given report is relatively modest. For instance, with AIS data there are 24 types of messages resulting in a total of 130 distinct attributes, and few of these attributes are used in multiple message types.

Each report's attribute is represented by an `AttributeValue` entity which has two main properties: the attribute itself and its value for the given report.

This model was selected mainly because each data type (AIS, LRIT, etc.) has its own attributes. Creating tables with columns matching all possible attributes of all MSARI data types would greatly limit the possibility to evolve the model and thus impact the extensibility of MSARI as described in requirement 3.1.6.b. Moreover, it would result in very large sparse tables (containing a lot of columns with large number of NULL values), which will impact the querying performance.

In order to keep attributes names (`AttributeName`) meaningful for MSARI users and administrators, the AIVDM vocabulary [23], a standard layered on top of NMEA 0183 and used for the AIS, was selected to describe attributes. In order to allow the model to gracefully evolve and ease querying, this standard should be kept for the addition of new data types. The latest complete list of attributes can be found in the MSARI system guide [14].

8.3 Physical Data Model

Figure 12 represents the physical representation of the conceptual model illustrated in Figure 11. This diagram shows the table names, column names, column data types, primary keys (key symbol on the left side the table) and foreign keys (represented by the arrows).

While the conceptual model focuses on simplicity and extensibility requirements (first two items in the section 8.1 list), the main challenge for the physical model design, i.e. the conceptual model implementation, is database scalability.

MSARI has to contain raw data, parsed data and metadata and the relations between them. Therefore, 1 GB of raw data input occupies more than 1 GB of disk space once acquired and stored by MSARI. If not designed adequately, the MSARI database size may explode after 1 year of data acquisition.

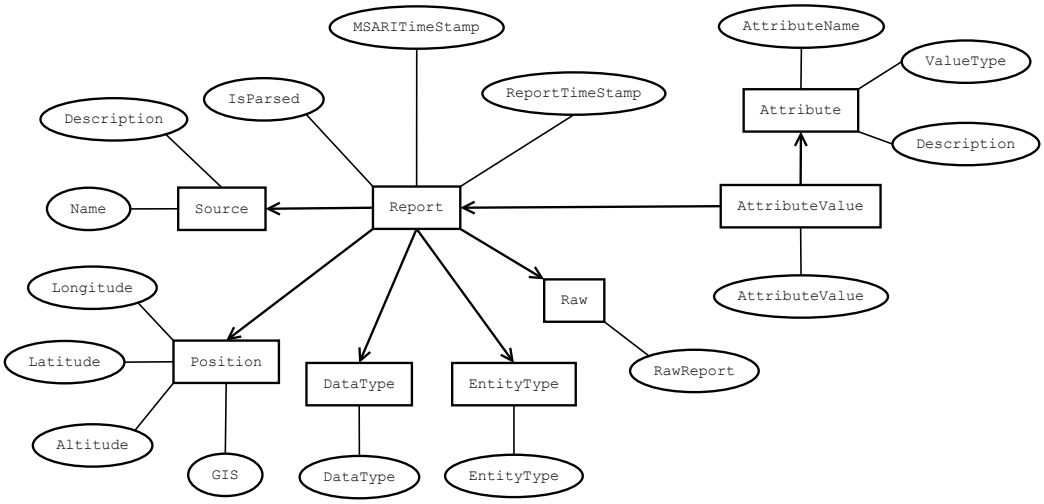

Figure 11: *Conceptual model of the MSARI database.*

Even though PostgreSQL has no limit to the maximum database size and can handle tables of size up to 32 TB [24], the query speed performance may be affected dramatically as the database grows over a certain threshold. The physical data model has to be designed such that the database is able to cope with such amounts of data. One of the challenges comes in minimizing the database footprint per AIS message (and other type of data). Moreover, even if the database footprint is minimized, the disk space consumption after 1 year is expected to exceed 5 TB. If not designed adequately, the MSARI database scalability may not be ensured when the number of rows in the database grows to that level.

In order to both reduce the space and speed up the query execution, three main design choices have been made and an optional one is proposed:

1. Partitioning the attribute values;
2. Indexing;
3. Column data types selection;
4. Limit the number of attributes to be stored.

The `AttributeValues` entity has been partitioned into 3 tables, based on the type of the value: int, text, double. Tests were made without partitioning, i.e. using a single table `attribute_values` with all attributes stored as text. The `attribute_values` table was growing very fast compared to other tables (took about 85% of the entire database disk space). The reason for this growth is there are several attributes for each report and the attribute values were stored as varying characters, while the majority of the attribute values are codes such as MMSI or double values such as course. So, storing everything as text

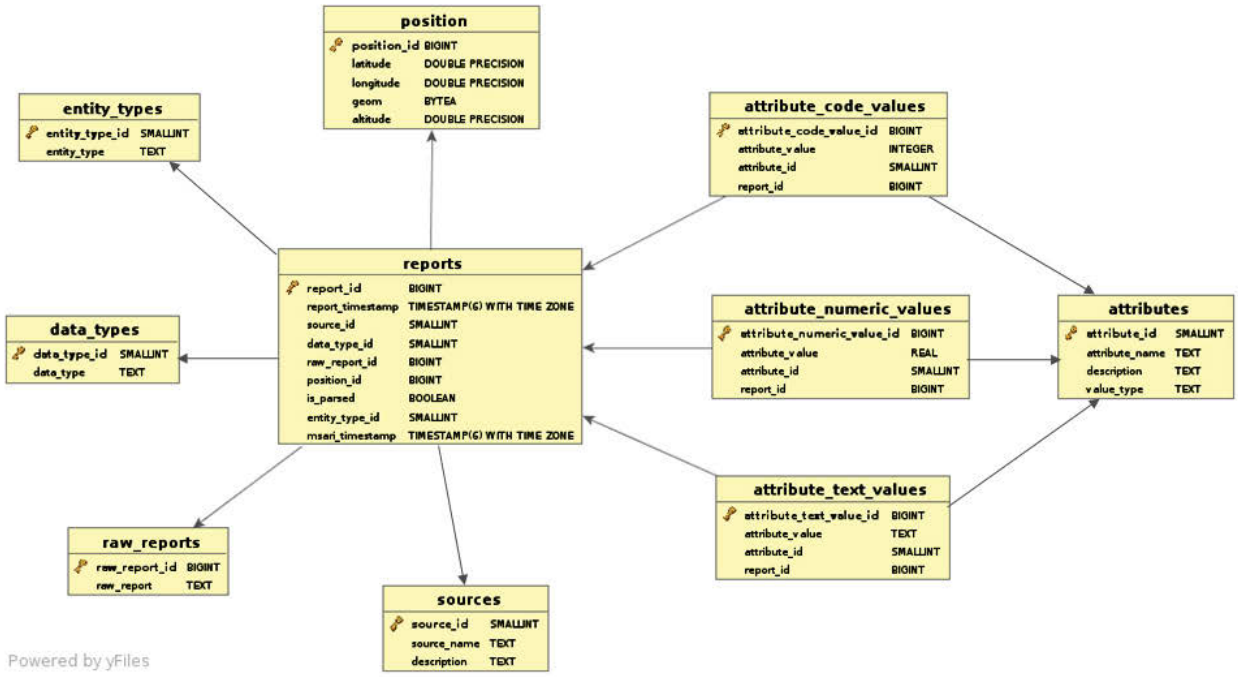

Figure 12: *Physical model of the MSARI database. Note that GIS are omitted of this diagram for clarity reasons.*

is not optimal and is costly in terms of storage (and thus query execution speed) on the medium-long term. For example, the MMSI number *538003407* occupies 13 bytes on disk when stored as text while it takes 4 bytes when stored as an integer. Partitioning the `attribute_values` table based on the attribute value types saved a lot of space and casting on the attribute value is no longer required. Moreover, as usual with partitioning, it provides an additional opportunity to optimize queries. It is also the common practice to segregate values based on data type for the EAV model.

Composite index for the tables `attribute_code_values`, `attribute_text_values` and `attribute_numeric_values` table on columns `attribute_id` and `attribute_value` were added. Because the attribute values tables are queried based on an attribute and its value, these indexes speed up the queries based on attribute values.

Finally, for each table we chose smallint or integer, instead of bigint when it was possible. For example, the number 100 stored as a smallint occupies 2 bytes on disk, while stored as integer and bigint it takes 4 bytes and 8 bytes respectively.

An additional design option would limit the space on disk of the MSARI database: limit the number of attributes to only the essential ones for data querying. Because all raw data is stored and available for each report, it is always possible to get back the value of all

attributes.

Maintenance operations are also part of the strategy to reduce table size on disk. Vacuuming (garbage collection), analyzing (collect statistics about a database) and reindexing (rebuild indexes) decrease significantly the size of a database (about 60%). Also, a list of PostgreSQL system parameters can be tuned based on the machine specifications. These maintenance operations and parameters tuning are described in the system guide [14].

8.3.1 Functions and Views

The EAV model ensures extensibility of the model, but it comes with a price: it increases the complexity of queries based on attribute values. As a compromise, the MSARI data model also includes SQL views and functions. Figure 13 illustrates the MSARI data model which includes two layers:

1. tables,
2. views and functions.

The user that is not proficient in SQL language should use the second layer to retrieve data. The SQL experienced user may also want to use the second layer functions because they include optimized queries, but can also work directly with the tables layer.

A view is a stored query accessible as a virtual table in a relational database. It is not physically materialized. Instead, the query is run every time the view is referenced in a query. These views aggregate the data stored in the tables in a meaningful way. For instance, the `vw_unparsed_reports` lists all reports that MSARI failed to parse by providing their ID, raw data, source name, data type and MSARI timestamp.

A function is a stored list of statements in SQL language (or PL/pgSQL, C or others) that may or may not take arguments and return either a table or a single value. The advantages of functions in the case of MSARI are twofold: masks the complexity from the user and shares standardized and optimized queries across the MSARI users.

This second advantage plays a very important role in performance. As explained in [25], since SQL is declarative, there are typically a large number of alternative ways to execute a given query, with widely varying performance.

There are mainly 2 ways to optimize a query:

1. at the input: filtering to minimize the number of rows/columns to look at to execute the query and
2. at the output: minimizing the number of rows and columns returned to only those that are necessary to the user.

Both strategies were used for MSARI.

A description of available views and functions is provided in the system guide [14].

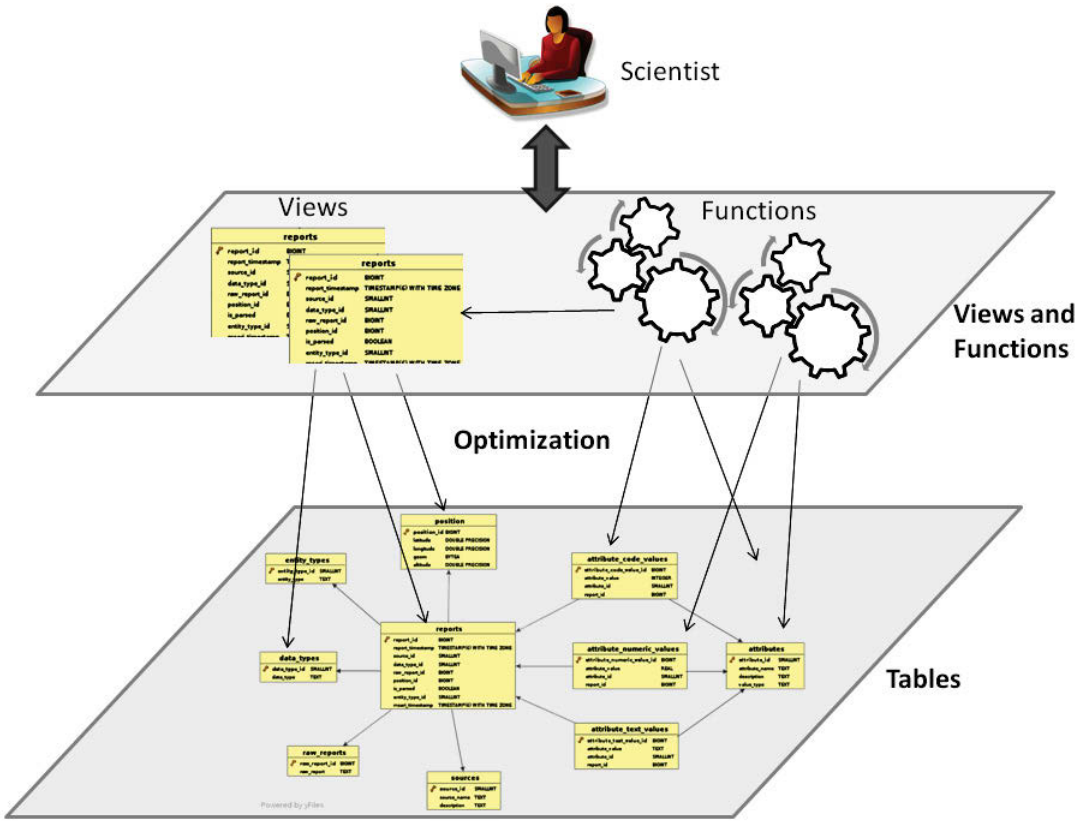

Figure 13: *Data model layers: tables layer and the views and functions layer. Users not familiar with SQL will access MSARI using the functions and possibly the views.*

8.3.2 Storing Strategy

As mentioned above, maintenance activities are required to decrease the size of the database and keep statistics up to date so that the PostgreSQL query optimizer can function at its best. Unfortunately, these maintenance operations are heavy on both CPU and disk, which affects query performance. Moreover, the continuous storing tasks may lead to index fragmentation which increases reindexing efforts.

An alternative strategy was developed for loading data: acquired data is stored temporarily in a staging database having the same structure as the MSARI database. Periodically, a bulk loading process is started to move the data from the staging database into the MSARI database. This strategy, also adopted in the PASTA-MARE project [19], is illustrated in Figure 14.

Data is continuously stored in the staging database by the acquisition component. At fixed intervals (e.g. 24 hours of storing), data in the staging database is inserted into the MSARI database (step 1 in Figure 14), then dropped from the staging database (step 2) and fi-

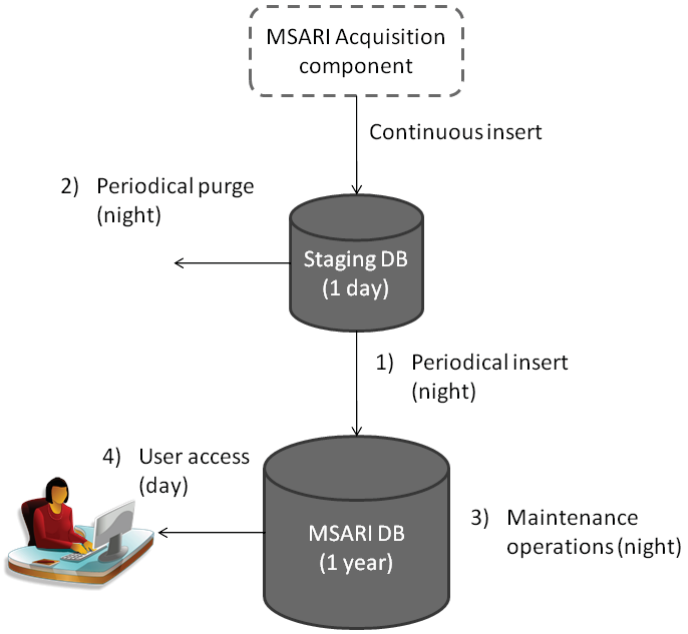

Figure 14: *Storing strategy: The data is continuously stored in a staging database and once a day, is inserted within the MSARI database (1) and then dropped (2). That way, storing and maintenance (3) of the MSARI database is performed only when users are the less likely to access (4) the database (e.g. during the night).*

nally, all maintenance procedures (vacuum, index rebuild and analyze) are performed on the MSARI database (step 3). This procedure is performed outside working hours (e.g. after 19h). That way, the users would interact with a static and maintained database.

The advantages of this strategy for MSARI are:

- Ensure optimal query performance;
- Ensure efficient insertion of data into the database;
- Avoid index fragmentation;
- Allow for maintaining the database periodically.

The inserts from the staging database to the main database are fast, because it is only pure SQL insert statements (both DB having the same structure).

8.3.2.1 Long Term Storing

After one year of storing, the MSARI DB size is expected to exceed 5 TB. This potentially large quantity of data gathered by MSARI may suggest multiple databases over the years are the best option. With a single database, query speed performance may be affected dramatically as the database grows over a certain threshold. A possible configuration is to have a database for each time period (e.g. one continuous year of data).

The choice of one database versus multiple databases depends mainly (but not only) on the characteristics of the data to be stored.

The advantages of multiple databases over a single one are:

1. Databases are smaller and produce faster queries.
2. Queries are simpler.
3. It minimizes the risk of one data set polluting another.
4. A single database could pose a performance bottleneck as it gets large (number of entities increase).
5. Easy data clean up as databases or entities are removed.

In any case, intensive testing will be required in order to make that decision.

9 User Interface Design

This section describes the user interface to access and maintain data. The following describes the characteristics of the user interface that allows data access and maintenance.

9.1 MSA Data Query Interface

The MSA Data Query interface provides the user with a graphical means for accessing data. It exposes the data access functionalities described in 7.4.

As described in 7.4.1, the proposed approach for the data access design is to test and compare the existing PostgreSQL/PostGIS front-end tools with MSARI data store and data. Since some MSARI users have little or no experience with SQL, the selected tool, or combination of, has to include a graphical query builder. If no query builder tools for PostgreSQL and PostGIS are found satisfactory for the MIS group member's needs, an SQL abstraction layer and a custom MSA data query GUI will be developed.

The open source front-end tool option is favoured over the custom GUI because of the following reasons:

1. a custom GUI does not provide the user with the full power of the database;
2. the modification of the data model (e.g. addition of a data type and its attributes) would require modifying the custom GUI.

Among the existing free front-end tools for PostgreSQL/PostGIS, pgAdmin III stands out. It has a graphical query builder, but it is not clear how robust the graphical query builder is for the MSARI users needs. As a compromise solution for the graphical query builder, it is possible to define and store SQL queries and execute them with a single key press. The query output can be exported in CSV format or just pasted from the output window. The query output can also be graphically visualized, enabling the user to find out how the query is parsed, optimized and executed. This feature can ease the SQL learning process for inexperienced users. Moreover, the GUI will remember all commands executed, the results from them and the execution time, until it is cleared by the user. This history will help the user to estimate the execution time for a given query. It also fully supports PostGIS and has a Shapefile loader plug-in.

Another very interesting alternative to pgAdmin III is DbVisualizer [26]. This commercial tool can be used for accessing a wide range of databases such as PostgreSQL, mySQL and Oracle. It has interesting functionalities such as charting, is intuitive and has a great look-and-feel.

Another option is to use MS Access as a front-end for PostgreSQL (for non spatial objects) in combination with Quantum GIS [27] for spatial objects [4].

4. MS Access does not support PostGIS spatial queries

All options cover requirements 3.2.1.a, 3.2.1.b, 3.2.1.c, 3.2.1.d and some of the nice-to-have options 3.2.1.e.

But since MS access is a proprietary software and pgAdmin III is the suggested tool for data quality management (see 7.3.2), the first option is favoured. However, the final decision will be based on tests results with the MSARI databases and users opinions.

In the unlikely case that a custom GUI has to be developed, it will be a web-based query interface allowing, as a minimum, data search with the following criteria (with logical operators AND/OR, multiple choices for each item and wildcards):

1. Ship: IDs such as MMSI, IMO, callsign, flag, name.
   – Text fields
2. Time: time intervals (e.g. May 12 - August 23 2011) and time periods (e.g. 1st week of each month for a given year).
   – Calendars
3. Source: source name and data type.
   – Check boxes
4. Position: bounding box (IN/OUT) and boundaries.
   – Text fields
5. SQL field
   – Text fields
6. Data format: Raw versus parsed data
   – Check boxes.

9.2 State Board

The state board is a graphical means to provide information about the state of MSARI. That user interface displays which data sources are up or down, data throughput, trends over time, users connected to MSARI, etc. In other words, it exposes the functionality of the performance maintenance component described in 7.3.1.

With the acquisition performance and data input being assessed based on the source meta-data, the pgAdmin III tool can be used for a quick awareness of the performance.

Moreover, there are open-source tools available for monitoring the states, memory and traffic of PostgreSQL databases. Amongst these, the dashboard pgwatch [28] is a good fit for MSARI [5]. Figure 15 is a screenshot of one of the multiple views of the pgwatch dashboard.

It supports monitoring of a variety of parameters:

5. Note that it supports only the latest versions of PostgreSQL - 9.0 and higher.

- Disk I/O;
- Cache hit rates over time and for tables/indexes;
- Checkpoints (timed vs. requested etc.);
- System activity;
- Open database connections;
- Active queries;
- Number of transactions (COMMIT, ROLLBACK, etc.);
- Optimizer information;
- Sequential scans;
- Index scans.

It provides a large array of features such as web browser display, easy configuration, large number of ready-to-use charts, dashboard for fast system checking, automated collection of statistics, integrated SQL worksheet. A custom dashboard will be configured for MSARI administration needs. The possibility to integrate the acquisition performance information will be investigated.

The pgAdmin III with the pgwatch dashboard cover all performance monitoring requirements 3.2.2.a.

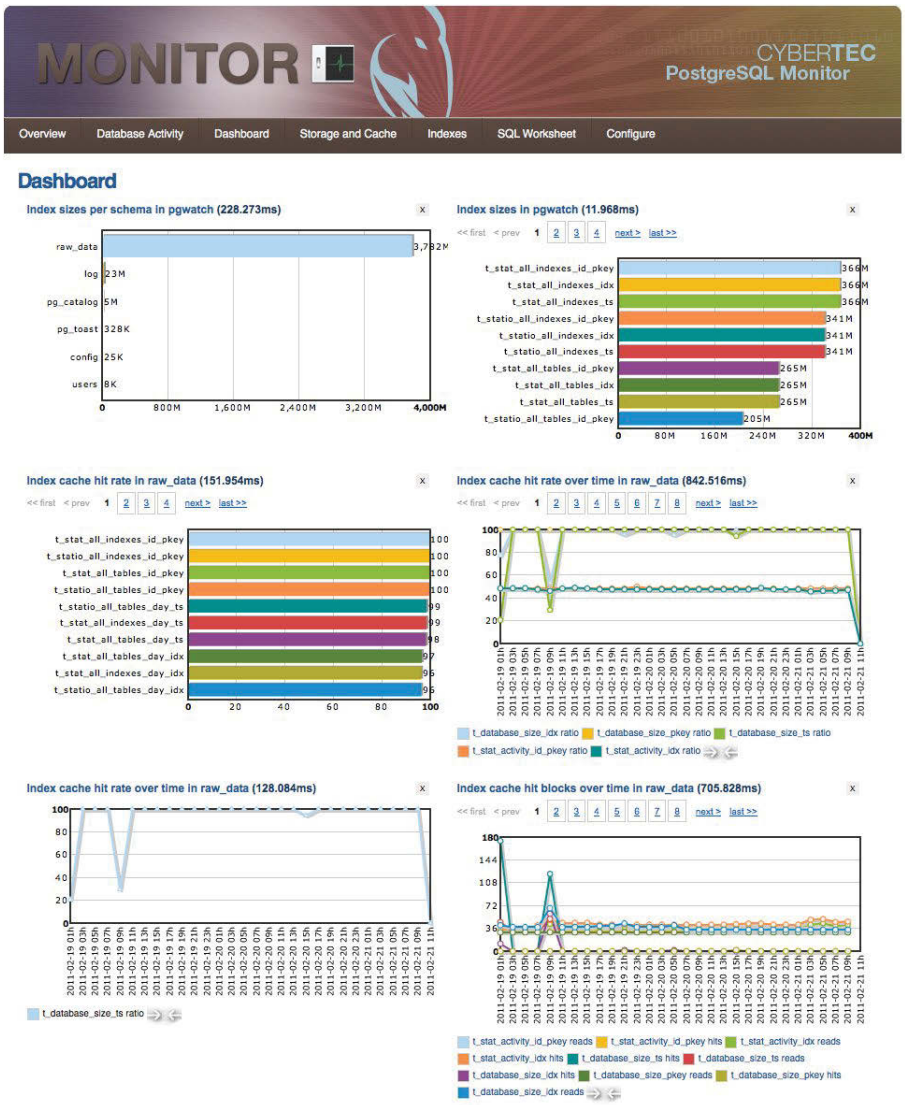

Figure 15: *One of the views of the pgwatch dashboard.*

10 Requirements Traceability

This section contains:

1. Traceability from each software unit identified in section 7 to the MSARI functional requirements allocated to it (from sections 3.1, 3.2 and 4.1).
2. Traceability from each MSARI functional requirement to the software units to which it is allocated.

Functional Component	**Software Units**	**Requirements**
Data Acquisition	Scheduler	3.1.1.a
	Source Client	3.1.1.a
	Parser	3.1.1.b
Data Storing	MSARI Databases (with data model)	3.1.2.a, 3.1.2.b, 3.1.2.c, 3.1.2.d, 4.1.a
	DBMS	3.2.2.b, 4.1.a, 4.1.b
	Database Manager	3.1.2.a, 3.1.2.c, 3.1.2.d
Data Maintenance	Performance Maintenance	3.1.3.d
	Data Quality Maintenance	3.1.3.a, 3.1.3.b, 3.1.3.c
Data Access	Data Access Service	3.1.5.a, 3.1.5.b, 3.1.5.c, 3.1.5.d, 3.1.5.e
Data Processing	Process Connector	3.1.4.a
User Interfaces	MSA Data Query Interface	3.2.1.a, 3.2.1.b, 3.2.1.c, 3.2.1.d, 3.2.1.e
	State Board	3.2.2.a

Table 2: *Software Units - Requirements Traceability Matrix*

Requirements	Software Units
3.1.1.a	Scheduler, Source Client
3.1.1.b	Parser
3.1.2.a	MSARI Databases, Database Manager
3.1.2.b	MSARI Databases
3.1.2.c	MSARI Databases, Database Manager
3.1.2.d	MSARI Databases, Database Manager
3.1.3.a	Data Quality Maintenance
3.1.3.b	Data Quality Maintenance
3.1.3.c	Data Quality Maintenance
3.1.3.d	Performance Maintenance
3.1.5.a	Data Access Service
3.1.5.b	Data Access Service
3.1.5.c	Data Access Service
3.1.5.d	Data Access Service
3.1.5.e	Data Access Service
3.1.4.a	Process Connector
3.1.6.a	Scheduler, Source Client, Parser
3.1.6.b	MSARI Databases (data model)
3.1.6.c	Process Connector
3.1.6.d	Data Access Service
3.2.1.a	MSA Data Query Interface
3.2.1.b	MSA Data Query Interface
3.2.1.c	MSA Data Query Interface
3.2.1.d	MSA Data Query Interface
3.2.1.e	MSA Data Query Interface
3.2.2.a	State Board
3.2.2.b	DBMS
4.1.a	MSARI Databases, DBMS
4.1.b	DBMS

Table 3: *Requirements - Software Units Traceability Matrix*

References

[1] (2011), MSSIS (online), John A. Volpe National Transportation Systems Center, https://mssis.volpe.dot.gov/Main/home/ (Access Date: 2011).

[2] (2011), Apache Camel (online), Apache Software Foundation, http://camel.apache.org/ (Access Date: 2012).

[3] (2011), Hibernate (online), JBoss Community, http://www.hibernate.org/ (Access Date: 2011).

[4] (2011), Hibernate Spatial (online), hibernatespatial.org, http://www.hibernatespatial.org/ (Access Date: 2011).

[5] (2011), Observer Pattern (online), Wikipedia, http://en.wikipedia.org/wiki/Observer_pattern (Access Date: 2011).

[6] (2011), Strategy Pattern (online), Wikipedia, http://en.wikipedia.org/wiki/Strategy_pattern (Access Date: 2011).

[7] (2011), PostgreSQL (online), PostgreSQL Global Development Group, http://www.postgresql.org/ (Access Date: 2011).

[8] (2011), MySQL (online), mysql.com, http://www.mysql.com/ (Access Date: 2011).

[9] (2011), PostGIS (online), OSGeo Project, http://postgis.refractions.net/ (Access Date: 2011).

[10] (2011), MySQL Spatial Extensions (online), mysql.com, http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html (Access Date: 2011).

[11] (2011), mongoDB (online), 10gen, http://www.mongodb.org/ (Access Date: 2011).

[12] (2011), pgAdmin - PostgreSQL Tools (online), pgadmin.org, http://www.pgadmin.org/ (Access Date: 2011).

[13] (2011), pgsnap (online), http://www.pgsnap.projects.postgresql.org (Access Date: 2011).

[14] St-Hilaire M.-O., Radulescu Dan and M., Mayrand (2013), Maritime Situational Awareness Research Infrastructure (MSARI): System Guide.

[15] Gingell, M. and Tremblay, D. (2009), Systems Design Document - MUSIC TDP, In support of the Multi-Sensor Integration in a Common Operating Environment, Fusion Test Bed Technology Demonstrator Project.

[16] (2011), NIEM (online), NIEM, http://www.niem.gov/ (Access Date: 2011).

[17] Isenor, A.W. and Spears, T.W. (2009), Utilizing Arc Marine Concepts for Designing a Geospatially Enabled Database to Support Rapid Environmental Assessment.

[18] Favre, S. and Eiden, G. (2009), Technical Note TN 1: Description of AIS project database - Preparatory Action for Assessment of the Capacity of Spaceborne AIS Receivers to Support EU Maritime Policy.

[19] Raunstrup, T. Vibe (2009), Technical Note TN 2. Description of AIS project database: Preparatory Action for Assessment of the Capacity of Spaceborne AIS Receivers to Support EU Maritime Policy.

[20] St-Hilaire M.-O., Mayrand M. and Isenor, A. W. (2011), Implicit Trust in a Data Model.

[21] (2012), EntityAttributeValueModel (online), Wikipedia, http://en.wikipedia.org/wiki/Entity-attribute-value_model (Access Date: 2012).

[22] (2011), Star Schema (online), Wikipedia, http://en.wikipedia.org/wiki/Star_schema (Access Date: 2011).

[23] (2011), AIVDM/AIVDO protocol decoding (online), Eric S. Raymond, http://catb.org/gpsd/AIVDM.html (Access Date: 2011).

[24] (2012), PostgreSQL About (online), PostgreSQL Global Development Group, http://www.postgresql.org/about/ (Access Date: 2012).

[25] (2012), Query Plan (online), Wikipedia, http://en.wikipedia.org/wiki/Query_plan (Access Date: 2012).

[26] (2012), DbVisualizer DbVisualizer (online), DbVis Software, http://www.dbvis.com/ (Access Date: 2012).

[27] (2011), QuantumGIS (online), OSGeo, http://www.qgis.org/ (Access Date: 2011).

[28] (2011), pgwatch (online), http://www.cybertec.at/en/postgresql_products/pgwatch-cybertec-enterprise-postgresql-monitor (Access Date: 2011).

[29] Smith, Gregory (2010), PostgreSQL 9.0 High Performance, Packt Publishing.

[30] Lapinski, Anna-Liesa S. and (2011), Anthony W. Isenor (2011), Estimating Reception Coverage Characteristics of AIS, *Journal of Navigation*, pp. 609–623.

[31] D., Radulescu (2012), AIS Indexer Development Report.

This page intentionally left blank.

Annex A: Modification to MSARI design: second development iteration

This annex covers the modifications made to MSARI design during the second phase of development performed under Call-Up 7 of contract W7707-115137.

A.1 Data Model Refactoring

The logical model, described in section 8.2, was slightly modified which reflected on the physical model (see section 8.3). Moreover, additional modifications were made to the physical model to improve performance. The latest version of the conceptual model and physical model are illustrated respectively in figures A.1 and A.2.

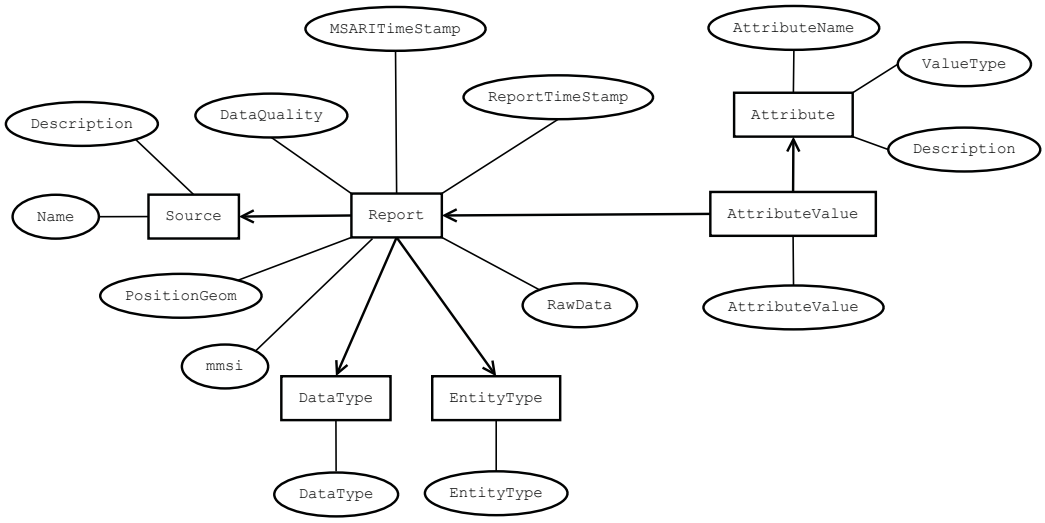

Figure A.1: *Latest conceptual model of the MSARI database.*

The following modifications were made:

1. Merge tables `raw_reports` and `position` to the table `reports`.
2. Partition more the attribute values.
3. For each attribute values table: use `report_id` and `attribute_id` as composite primary key.
4. Addition of MMSI in the `reports` table.
5. Remove all foreign keys.

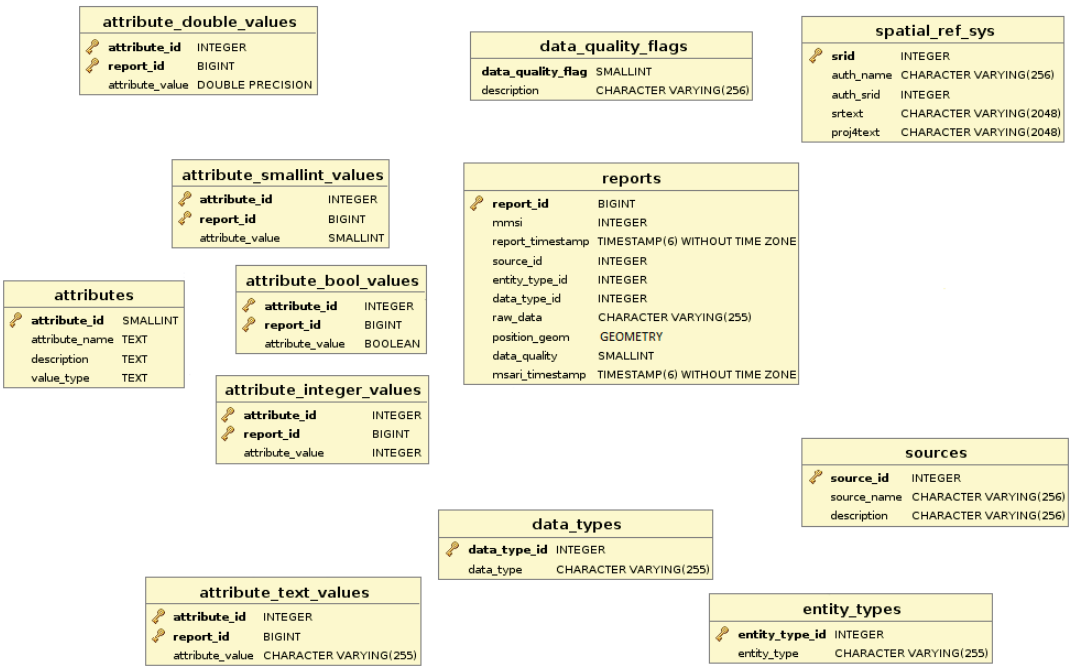

Figure A.2: *Latest physical model of the MSARI database.*

6. Replace the boolean flag `is_parsed` by a more complete data quality flag.

The first three modifications enabled savings of up to 38% in disk space and the speed of spatial queries execution was significantly improved. The fourth modification while not consistent with the MSARI source loosely-coupled design, greatly simplified maritime traffic queries. The fifth modification drastically sped up the acquisition process and deleting operation. Finaly, the sixth modification allowed the extension of the model with data quality information.

A.1.1 Table Merge

The content of tables `raw_reports` and `position` was merged to the `reports` table to save disk space and speed up spatial queries. For both tables, primary keys were stored as bigint (consuming 8 bytes of disk space). Merging the content of these tables to the main table removed the need for these primary keys. In large databases such MSARI, this kind of economy makes a considerable difference. Moreover, with the position in the `reports` table, there is no need to perform table joins. Since it is anticipated that scientists will often perform spatial queries, this modification removes the extra overhead associated with table joins and will make a difference on a daily basis.

A.1.2 Partitioning

Two attribute values table were added to the model: one for attribute values of type smallint and one for boolean values. Moreover, almost all of the attribute tables were renamed to better reflect the type of attribute values stored in the table:

Table name	**Attribute Value Type**	**Storage Size** (bytes)
`attribute_bool_values`	boolean (true/false)	1
`attribute_smallint_values`	smallint (-32768 to +32767)	2
`attribute_integer_values`	integer (-2147483648 to 2147483647)	4
`attribute_double_values`	real (6 decimal digits precision)	4
`attribute_text_values`	character varying(255)	More than 4

Table A.1: *Attribute values tables naming and content.*

A.1.3 Primary Key Strategy for Attribute Values Tables

This modification was the most important in terms of disk space. Each of the 5 attribute value tables (see Table A.1) had a primary key of type bigint occupying 8 bytes on disk. Because of the very large number of attributes stored by MSARI, these keys were occupying an important part of MSARI total disk space. The solution was to use `report_id` and `attribute_id` as a composite primary key.

The benefits of this change are threefold:

1. Save disk space.
2. No need to add extra indexes on `report_id` and `attribute_id`; and thus save disk space.
3. Speed up attribute-based queries.

A.1.4 Foreign Key Strategy

While the conceptual model has relationships between entities, the physical model does not contain any. Several tests were performed without any, with some, and with all of the foreign keys. It was found that the relation checks performed by PostgreSQL and Hibernate prior to inserting and deleting data, forced by the existence of foreign keys, was a performance bottleneck. And as the database size increases, this performance issue becomes more important.

Although it is good practice to have foreign keys in a database, a trade-off had to be made between integrity verification and performance. Because MSARI data throughput is already high (currently about 14 GB per day) and it may increase in the future, it was

estimated that the slowness in data storing and deleting processes would soon become problematic. Because MSARI will be mostly used by scientists to read data (instead of storing/deleting), data integrity will be maintained by the acquisition component.

A.1.5 Data Quality

In the first version of MSARI, the parser was filtering out all bogus data (including out-of-range) and non available data (see Section 7.1.3). For this second phase of development, it was decided to identify and store these data. While parsing the data, data quality is assessed based on the data source specifications. The data quality is then stored as part of the report's metadata.

In the table `reports`, the field `is_parsed` was replaced by `data_quality`, a bitmask which contains an integer value encoding the quality status of the data describing the report. A table called `data_quality_flags` was added to describe the quality flags and its content is listed in Table A.2.

For instance, a report with a `data_quality` value of 17 has a least one non-available attribute and a string attribute that was cleaned before being stored (e.g. callsign of @@@@@@@VE7SED@@@@@@@@ stored as VE7SED). The field can be queried using bit-wise AND operator & with a bit mask. For instance, the following query would retrieve all reports with a position out-of-range:
`SELECT * FROM reports WHERE (data_quality & 2) = 2`. Note that this flag does not provide which or how many attributes are concerned by the quality assessment. This granularity in quality assessment allows flexibility and is consistent with the source loosely coupled data model.

A.2 Hardware

The projected size of the MSARI DB and the aspired performance enforces new hardware requirements.

A.2.1 Requirements

The hardware is selected based on the following criteria:

1. Dedicated database server for the MSARI database
2. Dedicated acquisition server for receiving last 24 hours data from different sources. This data is then sent daily to the MSARI database to be updated.
3. Disk space predictions for one year of data (MSSIS and Exact Earth only), between 4 Terabytes (TB) to 6 TB depending on the fingerprint size (byte per report). This prediction does not take into account possible future traffic increases.

Data Quality Flag	Description
0	No error
1	Extra characters - Characters (e.g. @ and \) were removed before storing.
2	Position out of range - Range is defined in the source specs. (e.g. latitude outside [-90,91] or longitude outside [-180,181] for AIS source)
4	Attribute out of range - The range is defined by the source specs
8	Position not available - Default value defined in the source specs. (e.g. latitude = 91 and longitude = 181 for AIS source)
16	Attribute not available - Default value defined in the source specs (e.g. heading = 511 for AIS source)
32	Invalid message format - Message could not be parsed and is stored as raw format only
64	Invalid date/time format - Date could not be parsed and is stored as raw format only
128	Invalid attribute format - Attribute could not be parsed and is stored as raw format only
256	Invalid MMSI format - MMSI could either not be parsed (is stored as raw only) or is outside [7,9] digits and is stored as is

Table A.2: *Quality flags description.*

4. Relatively small number of users accessing MSARI (see section 4 requirement 4.1: shall support minimally the MIS group, which totals 6 users).
5. Typical database server criteria: robustness, redundancy and good I/O capability.
6. Redundant Array of Independent Disks (RAID) administration could be complicated, effort should be made to facilitate disk administration and more specifically, easily adding more space storage in the disk array.
7. To fit in budget requirement ($10K).

A.2.2 Hardware and configuration selection

Based on the above criteria, configuration and hardware choices are described below (see [29]):

1. **Dual Processor or Single Processor motherboard:** This question can be rephrased: do we need many Central Processing Unit (CPU) cores or faster CPU cores. Typical

dual processor motherboard are limited to a certain category of processors (ex: Intel Xeon) and chipsets. Each processor can have two to up to eight cores. Now, recall that each PosgreSQL query requires one core, so the number of cores provides an indication of the number of queries that can be processed at the same time. On the other hand, it is expected that these queries will be intensive and possibly long to process which indicate that a faster core should be used in that case. This last requirement would indicate choosing a single but fast processor but on the other hand, having to serve many users, the recommendation would be to use a state-of-the-art dual processor motherboard, but the fastest configuration possible. This configuration will be able to serve many users, and evolve gracefully to accept new ones.

The recommendation is to use Asus latest dual-board Z9PE-D16/2L for both servers. Also this motherboard can accommodate as much as 512 Gigabyte (GB) of Random-Access Memory (RAM) which should be useful to deal with future workload increases.

2. **CPU:** For reliability and performance, it is recommended to use Intel CPU instead of the cheaper AMD CPU, which fits with the above motherboard choice. The Z9PE-D16/2L motherboard requires Intel Xeon CPU from the E5-26XX family. These CPU can have 2 to 8 cores and their speeds range from 1.8 to 3.3 GHz [6].

   The first choice would be the two E5-2643 (4 cores at 3.3 GHz) according to the criteria above, but availability and prices force us to select two E5-2640 (6 cores at 2.5 GHz) for the main server and one E5-2640 for the acquisition server.

3. **Memory:** The Z9PE-D16 motherboard can hold as much as 512 GB of DDR3 RAM distributed in 16 slots. It is recommended to have enough memory to hold the full database but in this case, we already know that this will not be possible. Therefore, it is recommended to invest more on fast hard drives. We selected 32 GB ECC Kinston memory for the acquisition server and 64 GB for the main server.

   Consequently, the recommended amount of memory is 64 GB (1600 MHz) for the main server and 32 GB for acquisition server.

4. **Hard drive:** As specified above, we need very fast disks for very large databases. Two main standards are available: Serial Attached SCSI (SAS) and Serial ATA (SATA). SAS drives are usually preferred as being more reliable and faster (up to 15K RMP) than SATA but on the other hand, SAS drive capacity is often lower and more expensive than SATA drives. But recently, the gap between these two standards has decreased where hard drive companies started to produce high capacity drives with either SAS or SATA interface. For the purpose of this project, only the largest drives will be considered. The company Seagate provides the highest SAS (and SATA) at a reasonable price. The recommendation is to acquire six Seagate SATA ST33000650NS or SAS ST33000650SS (depending of their availability), both 3 TB, 6 GB/s and 7200RPM, set as RAID10 for a total of 6 TB or RAID5 for a total of 9

6. http://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors

TB. Seagate has just released recently the 4 TB model, if available for a reasonable price, it will be favored over the 3 TB model.

But because of a very high demand for large (SAS) disks, we had to purchase the only available equivalent, the very affordable 3 TB (SAS) Hitachi 7K3000. Four SAS Hitachi drives were purchased for the main server. Two 3 TB (SATA) Seagate were purchased for the acquisition server.

5. **RAID:** Because the database is expected to exceed the maximum storage of a single drive (commercially available), the use of an array of disks is mandatory. The option of using Network-Attached Storage (NAS) is interesting but too expensive for the allowed budget. Also, NAS administration, although simpler to use, is restricted to the hardware chosen and is less versatile than a custom one.

   We decided to try the ASUS PIKE RAID 2108 card compatible with our choose of motherboard. Because the benchmark performances were quite satisfactory, we did not investigate the possibility to use a Software RAID approach. Two RAID configuration were tested: RAID5 and RAID10, both using an array of four identical 3 TB SAS disks. The RAID5 configuration (provides a total of 9 TB of disk space and can continue to function after the failure of one disk although the performance is slightly diminished. The read performance is excellent but the write performance is much lower due to the parity calculation necessary for allowing disk redundancy. The RAID10 configuration provides a equally good performance for both read and write but only provides 6 TB of disk space instead of 9 TB.

	RAID5	**RAID10**
Read	510 MB/s	335 MB/s
Write	89 MB/s	308 MB/s

Table A.3: *Comparison of read/write performance between RAID5 and RAID10 based on benchmark tests.*

   The RAID5 configuration was chosen because for its larger capacity and due to the fact the MSARI database is mostly a ”read-only” database. The slower writing process will not be an issue as the transfer will be done over night. On the other hand, the users will benefit from a much faster read access for their queries. Currently there is no easy solution on how to gracefully increase a RAID array (neither for software or hardware) without blowing out the allowed hardware budget. The largest size solution (RAID5) was selected to compensate for the absence of this feature by allowing the storage of a larger set of data (9 TB).

6. **CD/DVD RW drive:** No specific requirement, any good model will do. The recommendation is to buy a Samsung CD/DVD RW drive.

7. **Case and power supply:** The motherboard is slightly bigger than the ATX factor form model, so a bigger and compatible EEB tower case is needed. We finally selected the Silverstone RV03B-WA and the power supply SST-ST85F-G from the

same company. The cables of the selected power supply were totally configurable which was very useful for dual processor motherboard and an array of disk.

8. **Operating system and software:** Linux is usually recommended for a server. The the most popular Linux server distributions are Centos, Debian and Ubuntu (64bit being mandatory for managing more than 4 GB of memory). In this case, Ubuntu 12.04 64bit is recommended. This recent version is a Long Term Support (LTS) version which will be supported for the next five years. For the file system, a journaling file system such as ext4, ZFS or XFS is recommended. Recent benchmark results favored ext4 or ZFS. As for the database, the latest PostgreSQL 9.2 64bit will be used for the MSARI server as well as Postgis 2.0.1 64bit for the GIS support.

The two Tables A.4 and A.5 show the details of the final selection for the hardware and links to the specifications of each part. The first server (the acquisition server) listens to the different socket sources and stores the information in its local instantiation of the MSARI database (called `msari_buffer`) and also make a local backup of the raw incoming information in a compressed format. During the night, the contents of the local database is then transferred and added to the second MSARI server database (called `msari`) which is solely dedicated to answer the user queries. Then, the transferred data is remove from the first server database in order to make room for more data to be transferred. Note that this transfer process does not disturbed the continuous reception of socket data. Figure A.3 illustrates the hardware configuration ant its connectivity with the DRDC network.

Item	Company	Model	Specification	Qty	Link
Motherboard	Asus	Z9PE-D16/2L	Dual Intel Xeon E5-2600, DDR3 1600, LGA2011, C602-A	1	link
CPU	Intel	BX80621E52630	Six Core, 2.3GHz, 15MB L3 Cache	1	link
Memory	Kingston	KVR16R11D4K4/32	4x8GB, DDR3, 1600MHz, 240pin,ECC	1	link
Hard drive	Seagate	ST33000650NS	SATA,3TB, 6Gb/s,7200rpm,64MB cache	2	link
CD/DVD RW	Lite-On	IHAS424-98 R	Internal, SATA, 1.5MB buffer	1	link
CPU fan	Dynatron	R17	CPU Socket 2011, Aluminum Fins, 4x Heat Pipes with H.C.C.	1	link
Power supply	Silverstone	SST-ST85F-G	850Watt, Modular, Active PFC supply, ATX 12Vv23	1	link
Case	Silverstone	Raven Series (RV03B-WA)	ATX Full Tower, SSI EEB compatible, 10x3.5 bay drives	1	link

Table A.4: *Configuration for the acquisition server.*

Item	Company	Model	Specification	Qty	Link
Motherboard	Asus	Z9PE-D16/2L	Dual Intel Xeon E5-2600, DDR3 1600, LGA 2011, C602-A	1	link
RAID card	Asus	PIKE 2108	8 ports RAID 0,1,5,6,10	1	link
CPU	Intel	BX80621E52630	Six Core, 2.3GHz, 15MB L3 Cache	2	link
Memory	Kingston	KVR16R11D4K4/32	4x8GB, DDR3, 1600MHz, 240pin,ECC	2	link
Hard drive	Hitachi	7K3000	SAS, 3TB, 6Gb/s,7200rpm,64MB cache	4	link
CD/DVD RW	Lite-On	IHAS424-98 R	Internal, SATA, 1.5MB buffer	1	link
CPU fan	Dynatron	R17	CPU Socket 2011, Aluminum Fins, 4x Heat Pipes with H.C.C.	2	link
Power supply	Silverstone	SST-ST85F-G	850Watt, Modular, Active PFC supply, ATX 12Vv23	1	link
Case	Silverstone	Raven Series (RV03B-WA)	ATX Full Tower, SSI EEB compatible, 10x3.5 bay drives	1	link

Table A.5: *Configuration for the main server.*

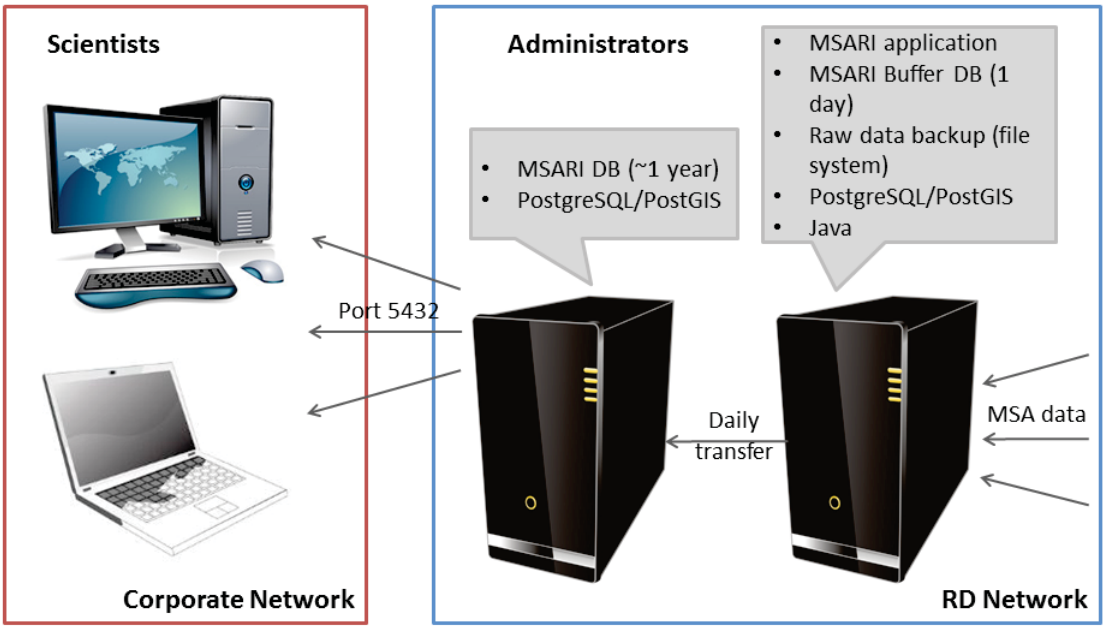

Figure A.3: *MSARI's hardware and connectivity with DRDC-A networks.*

A.3 Structural Modifications

Instead of having a congregation of parsers and source clients connected together in complex ways, MSARI now allows a single parser-source client connection.

In other words, many-to-one and one-to-many relationships described in 7.1.2 and 7.1.3 between source clients and parsers have been replaced in favor of one-to-one relationships. If new sources need to be listened to, a new MSARI instance can be created from the menu. From a high level perspective, not much has changed, but having this extra limitation simplifies the creation and communication between MSARI's internal components. Figure A.4 compares the new MSARI structure to the initial one.

A.4 Additional Features

Additional features were added to MSARI during the second phase of development. The following describes the most significant ones, in terms of design and functionalities.

A.4.1 Multi-Threaded Data Acquisition

One of the biggest hurdles was to increase the throughput of data to the database with minimal code alteration. The culprit in slowing the system down is Hibernate's interaction with

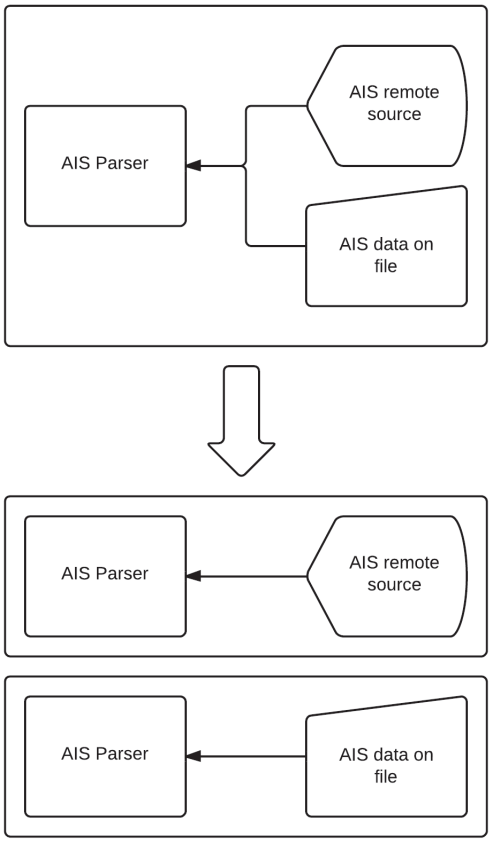

Figure A.4: *The illustration shows how a many-to-one relationship between source clients and a single parser type is handled in the new structure.*

the database. Since Hibernate is a black box outside of its configuration parameters, the solution to getting more data into the database was to multiply the channels that communicate with the database.

Hibernate's batch process was encapsulated in a thread that accepted 5000 reports at a time and started a session to transfer the data to the database. So instead of having one Hibernate session gather up a batch of MSARI reports, send it to the database, and wait for it to complete before commencing the next batch, MSARI now allows for the next batch to be sent as soon as it's ready simply by starting a new thread.

To simplify the code, a pool of a fixed number of threads is allocated that MSARI then starts as needed. It then waits for the pool of threads to terminate before continuing. If for example the number of threads was set to 3, then MSARI would start batch transfer sessions until all 3 threads were busy and then it would wait until they all completed before allowing 3 new transfers to start. Figure A.5 illustrates the data acquisition process with 3 threads.

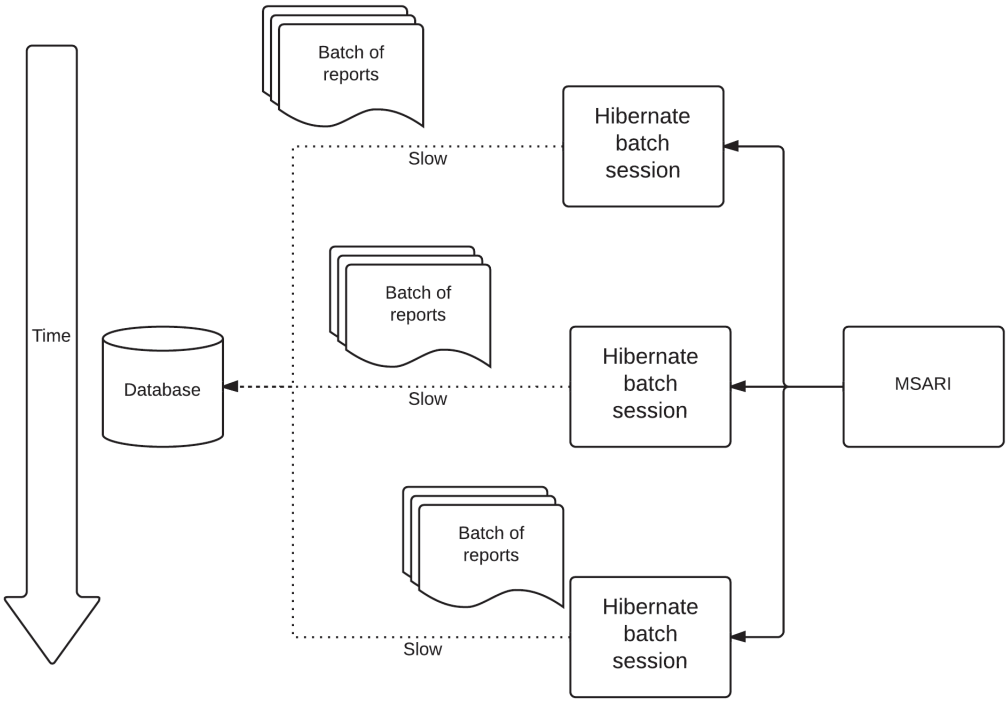

Figure A.5: *The illustration shows batches of reports being transfered to the database by three simultaneous sessions. Although each individual transfer session is slow, several simultaneous sessions lead to increased throughput.*

It is suggested that a maximum of 40 simultaneous sessions be allowed to run at once, beyond which tests have shown that throughput would diminish.

A.4.2 MSARI QI

The MSARI QI is an application that:
– allow the user to query the MSARI database using time windows, bounding box and source specification,
– output the results in a window and
– write the results in a file formatted for the AISIndexer (see [30] and [31] for more information about the AISIndexer).

The MSARI QI is a data processing algorithm where products are stored locally as a file, thus not stored back to MSARI. In the future, this application would be integrated to the AISIndexer.

The interface allowing to query MSARI heavily relies on Hibernate which reduces the required coding. In fact, in this case, the processing connector (see section 7.5.1) is Hibernate. The MSARI QI is a first step towards the completion of the MSARI processing component.

Figure A.6 is a screen capture of the MSARI QI Graphical User Interface (GUI).

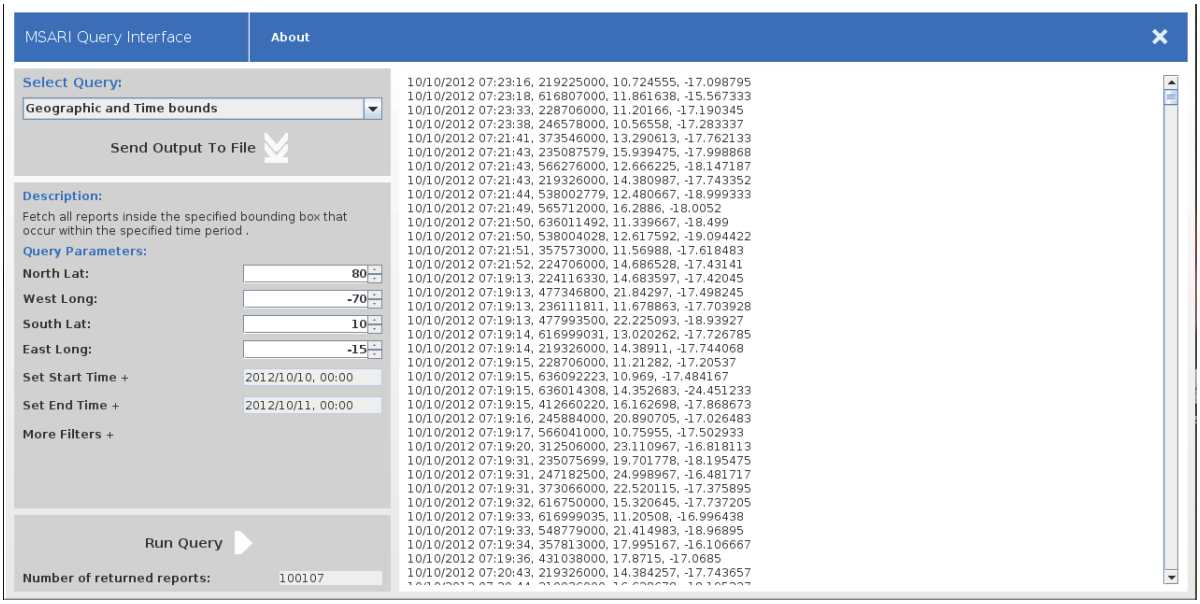

Figure A.6: *MSARI QI user interface.*

A.4.3 Enhanced AIS Decoder

The AIS decoder used is the java version of Brian Lane's decoder [7], licensed under Berkeley Software Distribution (BSD). The decoder was greatly modified along MSARI development. The resulting decoder includes message types 25 to 27, is fully adapted to space-based AIS messages and is more robust to different kinds of source specific timestamps. Also, it now decodes messages with up to 9 sequences for both shore and space-based AIS. This AIS decoder is now very robust and could be re-used for other applications at DRDC.

A.5 Testing

MSARI, with the described hardware, configuration, software and data model, was tested using a data set of 9 months of AIS data (from Exact Earth and MSSIS). The data acquisition process was tested using 3 sockets with a high data daily throughput (largely exceeding the expected data throughput at DRDC). The resulting MSARI database occupied about 4.2 TB on disk. It was found that the execution time of typical queries [8], following best practices, was satisfactory.

7. `https://github.com/bcl/aisparser`
8. See [14] for use cases queries.

List of symbols/abbreviations/acronyms/initialisms

ADS-B	Automatic Dependent Surveillance-Broadcast
AIS	Automatic Identification System
API	Application Programming Interface
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CSV	Comma-Separated Values
DB	Database
DBMS	Database Management System
DRDC	Defence Research and Development Canada
EAV	Entity-Attribute-Value
FTP	File Transfer Protocol
GB	Gigabyte
GIS	Geographic Information System
GUI	Graphical User Interface
HTML	HyperText Markup Language
ID	Identification Number
IMO	International Maritime Organization
JDBC	Java Database Connectivity
KML	Keyhole Markup Language
LRIT	Long Range Identification and Tracking
MDA	Maritime Domain Awareness
MIKM	Maritime Information and Knowledge Management
MIS	Maritime Information Support
MMSI	Maritime Mobile Service Identity
MSA	Maritime Situational Awareness
MSARI	Maritime Situational Awareness Research Infrastructure
MSSIS	Maritime Safety and Security Information System
NAS	Network-Attached Storage
NIEM	National Information Exchange Model
ORM	Object-Relational Mapping

OS	Operating System
QI	Query Interface
RAID	Redundant Array of Independent Disks
RAM	Random-Access Memory
RDBMS	Relational Database Management System
REA	Rapid Environmental Assessment
RJOC	Regional Joint Operations Center
SODA	Service Oriented Development Architecture
SQL	Structured Query Language
TB	Terabytes
UI	User Interface