European Maritime Safety Agency

Appendix 3
to Tender Specifications

Guidelines on implementation of web services and information exchange

EMSA

Introduction and overview

As the Safenet Ecosystem is and will be composed of services implemented by many different applications that today do not always collaborate the SSN Ecosutem ICT architecture group considers there is a need to harmonise the architectural styles within these applications. One of the aspects of architectural style is how services/applications exchange data internally or with external system. We chose to address this subject first as the interchange of information has the greatest and most immediate impact on all applications composing the SSN ecosystem.

This document tries to give a brief overview of the potential ways to exchange data between systems, give pros and cons of each and provide a guideline for when to use what integration style.

Methods of exchanging data between applications

Integration style ***Applicability***

Integration style	*Applicability*
Simple Object Access Protocol (SOAP) web services Is a protocol, is strongly typed and has a strict specification. SOAP is not limited to HTTP (e.g. JMS or SMTP may be used a transport mechanism). End-to-end security is supported through WS-* specifications, in contrast to REST where federated security (e.g. ADFS) is still work in progress (e.g. OpenID). SOAP has support for distributed, two-phase commit transactions, using WS-Atomic Transactions. There are many different WS-* specifications, multiple WS-* can address the same issue, so it is not always clear when to use which one and a particular WS-* spec could not be supported by all vendors. In limited cases, small differences in implementation between infrastructure vendors can cause interoperability problems. SOAP will uses interfaces and named operations to expose business logic as opposed to REST which exposes resources. SOAP has a set of standard specifications. WS-Security is the specification for security in the implementation. It is a detailed standard providing rules for security in application implementation. Like this we have separate specifications for messaging, transactions, etc. Unlike SOAP, REST does not have dedicated concepts for each of these.	*Recommended integration style for API style integration if there needs to be a tight control over the interface. This is most applicable when integrating with parties outside of EMSA or when contract negotiation will be applicable on changes to the interface.* *SOAP will also be the preferred integration style if additional functionality such as sending replies to different endpoints, asynchronous invocations, transactionality or other WS-* features are required.* *If combined with using JMS for message exchange it can provide further decoupling of applications and increased reliability and scalability.*
REST web services (XML) REST stands for Representational State Transfer and is not a protocol but rather an architectural style. Due to this, more attention needs to be paid to the quality of the implementation as opposed to web services using SOAP. Specifically the API design (REST URIs need to be resources, not methods), proper usage of HTTP verbs (GET to query resources, POST to create a new resource, PUT to update a resource and DELETE to delete a resource), discoverability (e.g. messages should contain links to next action(s) to be performed) and re-use of standard HTTP/web technologies (e.g. use HTTP authentication instead of implementing an own custom authentication protocol for the service, allow for caching of GET requests, …). For web services providing a business method (e.g. performing a calculation based on inputs) SOAP will usually be a better match. REST on the other	*Recommended integration style for API style integration when EMSA has a firm control over the interface and changes can be agreed on a less formal level (i.e. a simple agreement by the CAP will be sufficient).* *Using XML as the message format is recommended rather than using JSON format as there is better tool support for validation and transformation of messages. Also no standard exists for documenting or specifying the content of a JSON*

Integration style	***Applicability***
hand may be a more natural fit for exposing information (resources). REST does not define a contract for the service. Usage of the services will rely on documentation provided to the developer and validation of implementation of both client and server will be harder as opposed to SOAP web services. Web Application Description Language (WADL) is an attempt to formally define the API exposed by a REST based web service, but has not yet been standardised by W3C. Nevertheless all REST implementations at EMSA should deliver (and keep up to date) a WADL document describing the API as well as a human readable API user guide. • Whole of the web works based on REST style architecture. Consider a shared resource repository and consumers access the resources. • REST messages should be self-contained and should help consumer in controlling the interaction between provider and consumer(example, links in message to decide the next course of action). But SOAP doesn't has any such requirements. • REST does not enforces message format as XML or JSON or etc. But SOAP is XML based message protocol. • REST follows stateless model. SOAP has specifications for stateful implementation as well.	*message.*

Integration style	***Applicability***
REST Web services (JSON) All recommendations and cautions mentioned for REST Web Services (XML) apply here as well. Additionally, JSON format has more limited tool support, e.g. no validation of the message against DTD or Schema, no standards for message transformation as opposed to XSLT and XQuery).	*Allowed integration style for API style integration when EMSA has a firm control over the interface and changes can be agreed on a less formal level (i.e. a simple agreement by the CAP will be sufficient). Can be used instead of the XML message format if either* *1. interface will be consumed by a web user interface* *2. message size is very important*
Java Messaging Services (JMS) JMS can be used to achieve a loose coupling between applications exchanging data, allows for good scalability and reliable message delivery by relying on the underlying messaging infrastructure. The drawbacks are the additional complexity in setting up and configuring the messaging infrastructure.	*Recommended integration style for event driven data exchange.*
Shared data base tables Creates a tight coupling between applications and should be avoided if possible.	*In exceptional cases this may be an acceptable method of sharing information; however the reasons for choosing this integration method should be presented to the SSN Ecosystem Architecture Group which will then provide a recommendation to the ICT Steering Group.*
Remote procedure calls (RPC), CORBA, … These integration methods provide integration based on an API to be called by the client application. Whilst these may have performance benefits, especially if both applications run within the same machine or VM, it also reduces interoperability, e.g. both applications will usually need to be	*In exceptional cases this may be an acceptable method of sharing information; however the reasons for choosing this integration method should be presented to the*

Integration style	*Applicability*
implemented in the same technology (RPC) or use proprietary infrastructure (CORBA).	*SSN Ecosystem architecture which will then provide a recommendation to the ICT Steering Group.*

Message formats

Generally, there are 4 options when defining the message format for exchange of data between applications: XML, JSON, CSV or Binary objects.

eXtensible Markup Language (XML)

Using XML for exchanging data has the following main benefits:

- Human readable (useful for debugging, problem resolution).
- Supported by all major programming language and platforms.
- Commonly available developer skill.
- Message format is well defined. Strong support for message specification through either XSD (preferred) or DTD. This allows both validation (at run time) and documentation using a standard syntax.
- Cross platform / technology independent, e.g. 1 service implemented in PHP and running on a Windows Server can exchange messages with a Java application hosted on a Unix server.
- Good tool support: schema editors, validator, transformation, etc.

Disadvantages:

- XML has a major disadvantage that it can be very verbose and therefore may be less suitable for exchange short messages at a very high rate.

When defining an XML format, the best practice at EMSA should be to be as explicit as possible. This will enhance the understanding of the interface and reduce implementation errors and complexity. On the other hand, being explicit may require interface version to be updated more frequently. Yet, such updates will likely require changes or at least testing for compatibility with all of the service's client, so forcing an explicit version upgrade should be considered an advantage.

Applicability

Should be the preferred choice when exchanging large datasets at a low rate.
Is also the recommended format when exchanging data with 3th parties as the possibilities for validation of the data and format are more advanced and more well-known than for the other formats described in this document.

JavaScript Object Notation (JSON)

Using JSON for exchanging data has the following main benefits:

- Human readable (useful for debugging, problem resolution).
- Commonly available developer skill.
- Cross platform / technology independent, e.g. 1 service implemented in PHP and running on a Windows Server can exchange messages with a Java application hosted on a Unix server.
- Compact compared to XML, however still much more verbose than other options described below.
- Especially useful for exchanging data between the back-end services and the Web User Interface as JSON can be natively handled by all browser supporting JavaScript.

Applicability

Should only be used for providing data that will be displayed, directly, in a web browser. I.e. this format should only be used for services invoked by a graphical user interface. Even in this case, whenever data is exchanged in JSON format, both service client and service provider contracts need to be sufficiently flexible to allow dealing with incompatible interpretations of the message format.

Disadvantages:

- The main draw-back of JSON is the lack of standards and tool support. Especially defining and validating the contents of a JSON message is limited to humanly readable documentation only, excluding the use of commonly available validators, and thus may suffer different interpretation between the service provider and service client.

Comma separated values (CSV)

Data can be exchange as Comma Seperated Value filed. This is often useful when data will be generated manually, example as an export from an Excel file.

Advantages:

- Can be exported from MS Excel or other spreadsheet
- May be the most suitable option when interacting with a less technically competent 3th party.

Disadvantages:

- No standards for defining the message (e.g. various field delimiters are possible)
- Lack of tool support, e.g. validators and transformation

Applicability

Should only be used for exchanging datasets at a low rate with external parties that do not have the capacity to support one of the other formats defined in this document.

Binary protocols

Many different options exist. Below some of the most appropriate to EMSA are discussed.

Coherence Portable Object Format (POF)

Coherence has a proprietary data format which is designed to be compact and minimise resource usage.

Advantages:

- Small message size, i.e. very suitable for exchanging messages at high rate over the network
- Limited resource usage, e.g. CPU

Disadvantages:

- Proprietary format which requires the use of Oracle / Tangosol libraries
- Limited language support (only Java, .Net and C++)
- Not humanly readable
- POF requires the developer to implement routines in order to serialize and deserialize the objects. The (de)serialisation code will need to be provided as a library to all service clients.

Protocol Buffers

This was originally developed at Google and has been open sourced. As compared to POF it has similar goals for minimising resource usage (CPU and network) whilst also providing support for a larger number of languages. Additionally, usage does not require any licensing.

Advantages:

- Small message size, i.e. very suitable for exchanging messages at high rate over the network
- Limited resource usage, e.g. CPU

Applicability

Should be the preferred choice when exchanging data at a (very) high rate between EMSA maritime applications or between components within the same 1 application.

It should never be used for exchanging data with 3th parties.

The preferred binary protocol to be used within EMSA will be Protocol Buffers,

- Navigating an object tree can be easier than equivalent XML
- Message format is well defined

Disadvantages:

- Future / backward compatibility can be more tricky compared to XML
- Not humanly readable
- No tools for transforming between message formats or versions

Thrift

Is comparable to Protocol Buffers. The goals and advantages are very similar. Protocol Buffers are already used in the IMDatE application, therefore we recommend using only Protocol Buffers at EMSA as this will improve interoperability and limit proliferation of similar but different technologies.

Java serialized objects

Advantages:

- Out-of-the-box java feature

Disadvantages:

- No support for exchanging data with other programming languages
- Dependent on the version and implementation of the JDK.
- Not possible to maintain backwards compatibility other than through maintaining a separate implementation for older versions.

Service versioning

All services shall be versioned. The version shall include both a major and a minor version number. Optionally and extra number can be added to these 2 obligatory numbers, representing for example change due to emergency patches or minor bug fixes.

The major number shall be increased every time (and only if) there is a non-backward compatible change to the service interface, i.e. whenever the service's clients will require to be modified in order to still be able to make use of the latest version of the service.

The minor number shall be increased every time there are backwards compatible changes to the service interface. E.g. this could be the addition of extra methods, changes in responses or requests that do not necessary impact (all of) the clients.

If used, the optional third versioning number will be increased whenever there is a change in the service implementation but no change in the service interface. This version number will be used for internal purposes of the service only and should be completely transparent to any service clients.

For a more detailed explanation of the versioning scheme, please visit:
http://semver.org/spec/v1.0.0.html

Backwards compatibility

All services shall maintain compatibility for at least 1 major version[1].

All service owners shall be required to publish and keep up to date the planning for current and future versions of the service. This information needs to contain at least:

- Summary of the impact of planned changes. Especially any changes to the interface need to be clearly identified.
- Planned availability dates of the service (version) for each EMSA environment.
- Planned end of life date for the service (version) for each EMSA environment.

When a service is provided by a web server, the URI for the service shall include the major and minor version of the service as defined above. It shall never include the 3th, optional version number.

ICD Documentation

All services shall have both a human readable and a machine readable (where a technical standard exists).

SOAP webservices

SOAP webservices shall be defined by a WSDL document. The namespace of the WSDL document shall be:

- http://schemas.emsa.europa.eu/[ServiceName]/[major-version.minor-version].

Where possible, schema definitions for elements shall be reused between the services provided by EMSA. The namespace for element definitions shall be:

- http://schemas.emsa.europa.eu/[Entity]/[major-version.minor-version].

All schemas shall be published to a common repository and be publicly accessible at the URL corresponding to the namespace definition.

JSON webservices

All JSON webservices using XML message shall be defined by a WADL document.

For JSON services using JSON, there currently doesn't appear to be a commonly used interface specification. One possibility[2] might be WADL in combination with JSON Schema[3].

[1] This requirement can only be waived with the explicit agreement of all current or planned users of the service.
[2] http://kingsfleet.blogspot.pt/2012/11/json-schema-in-wadl.html
[3] http://json-schema.org/

Service clients

All systems / applications that rely or plan to rely on a service shall register their usage of the service in the Service Catalogue.

Service owners will exclusively use the Service Catalogue to contact parties that need to be informed of any changes, problems or planned maintenance activities. It therefor follows that unregistered clients of the service will not be informed of changes that may impact their application.