**RECONSURVE – A Reconfigurable Surveillance System with Smart Sensors and Communication**

*D9.2 – Current State of the Art in Standards*

Responsible Organization: Software Research and Development and Consultancy Ltd.

Due Date: 30.06.2011

Contents

[1 Table of Figures and Tables 4](#_Toc300060148)

[2 Executive Summary 5](#_Toc300060149)

[3 Introduction 7](#_Toc300060150)

[4 Scope 7](#_Toc300060151)

[5 Situational Awareness Standards 8](#_Toc300060152)

[5.1 Automatic Identification System (AIS) 8](#_Toc300060153)

[5.1.1 Applications 8](#_Toc300060154)

[5.1.2 Mechanism 11](#_Toc300060155)

[5.2 OGC Sensor Web Enablement (SWE) 13](#_Toc300060156)

[5.2.1 Sensor Markup Language (SensorML) 15](#_Toc300060157)

[5.2.2 Observations and Measurements (O&M) 18](#_Toc300060158)

[5.2.3 Sensor Observation Service (SOS) 19](#_Toc300060159)

[5.2.4 Sensor Planning Service (SPS) 20](#_Toc300060160)

[5.2.5 Sensor Alert Service (SAS) 22](#_Toc300060161)

[5.2.6 Web Notification Service (WNS) 23](#_Toc300060162)

[5.2.7 SWE Services Usages 25](#_Toc300060163)

[5.3 Description Logics Reasoners 27](#_Toc300060164)

[5.3.1 RULE ENGINES AND RULE LANGUAGES 28](#_Toc300060165)

[6 Interoperability Standards 31](#_Toc300060166)

[6.1 Joint Consultation, Command and Control Information Exchange Data Model (JC3IEDM) 31](#_Toc300060167)

[6.1.1 Overview of the Data Model 32](#_Toc300060168)

[6.2 SEMANTIC WEB TECHNOLOGIES 49](#_Toc300060169)

[6.2.1 Semantic Web 49](#_Toc300060170)

[6.2.2 Ontology 51](#_Toc300060171)

[6.2.3 Web Ontology Language (OWL) 52](#_Toc300060172)

[6.3 Web Services 57](#_Toc300060173)

[6.3.1 Web Service Architecture 64](#_Toc300060174)

[6.3.2 Web Service Protocol Stack 64](#_Toc300060175)

[6.3.3 Web Service Description Language (WSDL) 66](#_Toc300060176)

[6.3.4 Simple Object Access Protocol (SOAP) 68](#_Toc300060177)

[6.3.5 Universal Description Discovery and Integration (UDDI) 70](#_Toc300060178)

[6.4 DATA DISTRIBUTION SERVICE (DDS) 79](#_Toc300060179)

[6.4.1 Publish-Subscribe Mechanism Used By DDS 81](#_Toc300060180)

[6.4.2 Data-Centric Communication 81](#_Toc300060181)

[6.4.3 Data Distribution Service Components 81](#_Toc300060182)

[6.4.4 Topic and Topic Keys 86](#_Toc300060183)

[6.4.5 Multi Topics and Content Filtered Topics 87](#_Toc300060184)

[6.4.6 Quality of Service Parameters 87](#_Toc300060185)

[6.4.7 OpenDDS 88](#_Toc300060186)

[6.5 UN/CEFACT UNQUALIFIED DATA TYPES 90](#_Toc300060187)

[6.6 OASIS Common Alerting Protocol (CAP) 92](#_Toc300060188)

[6.6.1 History 93](#_Toc300060189)

[6.6.2 Applications of CAP message 93](#_Toc300060190)

[6.6.3 Structure of the CAP Alert Message 94](#_Toc300060191)

[7 Conclusion 96](#_Toc300060192)

# Table of Figures and Tables

[Figure 1 System Architecture 5](#_Toc300060193)

[Figure 2 Sample SensorML description 18](#_Toc300060194)

[Figure 3 Sample Observation 19](#_Toc300060195)

[Figure 4 Sensor Data Consumer Sequence Diagram 20](#_Toc300060196)

[Figure 5 SPS Interaction Diagram 22](#_Toc300060197)

[Figure 6 SAS Communication Diagram 23](#_Toc300060198)

[Figure 7 WNS registration and notification sample 24](#_Toc300060199)

[Figure 8 SOS and O&M usage 25](#_Toc300060200)

[Figure 9 SOS, SPS, WNS and O&M usage 26](#_Toc300060201)

[Figure 10 SAS and WNS Usage 27](#_Toc300060202)

[Figure 11. Independent Entities for Creating the Data Specification 38](#_Toc300060203)

[Figure 12. First Level Subtyping of OBJECT-TYPE and OBJECT-ITEM 40](#_Toc300060204)

[Figure 13. Entity-Level View of OBJECT-TYPE Subtype Tree 43](#_Toc300060205)

[Figure 14. Entity-Level View of OBJECT-ITEM Subtype Tree 45](#_Toc300060206)

[Figure 15. Plans and Orders Structure Shown at Entity Level 47](#_Toc300060207)

[Figure 16. Basic ACTION Structure 48](#_Toc300060208)

[Figure 17 A Basic Web Service 62](#_Toc300060209)

[Figure 18 XML Messaging for Web Services 63](#_Toc300060210)

[Figure 19 Web Services Role 64](#_Toc300060211)

[Figure 20 Web Service Protocol Stack 65](#_Toc300060212)

[Figure 21 WSDL Elements 68](#_Toc300060213)

[Figure 22 SOAP Message Structure 69](#_Toc300060214)

[Figure 23 UDDI Core Types 72](#_Toc300060215)

[Figure 24 Example Business Registration Template 78](#_Toc300060216)

[Figure 25 An instance for the message template in Figure 24 79](#_Toc300060217)

[Figure 26 The structure of Data Distribution Service 80](#_Toc300060218)

[Figure 27 Layered View of Distribution Service 82](#_Toc300060219)

[Figure 28 DCPS Entities 83](#_Toc300060220)

[Figure 29 Single Domain System 83](#_Toc300060221)

[Figure 30 Multiple Domain System 84](#_Toc300060222)

[Figure 31 Publish Data 85](#_Toc300060223)

[Figure 32 Subscribing Data 86](#_Toc300060224)

[Figure 33 Pluggable Transport Framework 89](#_Toc300060225)

[Figure 34 Document Object Model 96](#_Toc300060226)

[Table 1. Independent Entities and Their Roles 34](#_Toc300060227)

[Table 2. Definition of First-Level Subtypes 40](#_Toc300060228)

[Table 3 Six core class descriptions and their definitions 53](#_Toc300060229)

[Table 4 Unqualified Data Types 90](#_Toc300060230)

# Executive Summary

The RECONSURVE (A Reconfigurable Surveillance System with Smart Sensors and Communication) project has been motivated by and aims to address the need to control the rapidly increasing number and complexity of maritime surveillance issues such as illegal immigration especially using small vessels, interoperability between heterogeneous systems, automated cost-effective and efficient decision support.

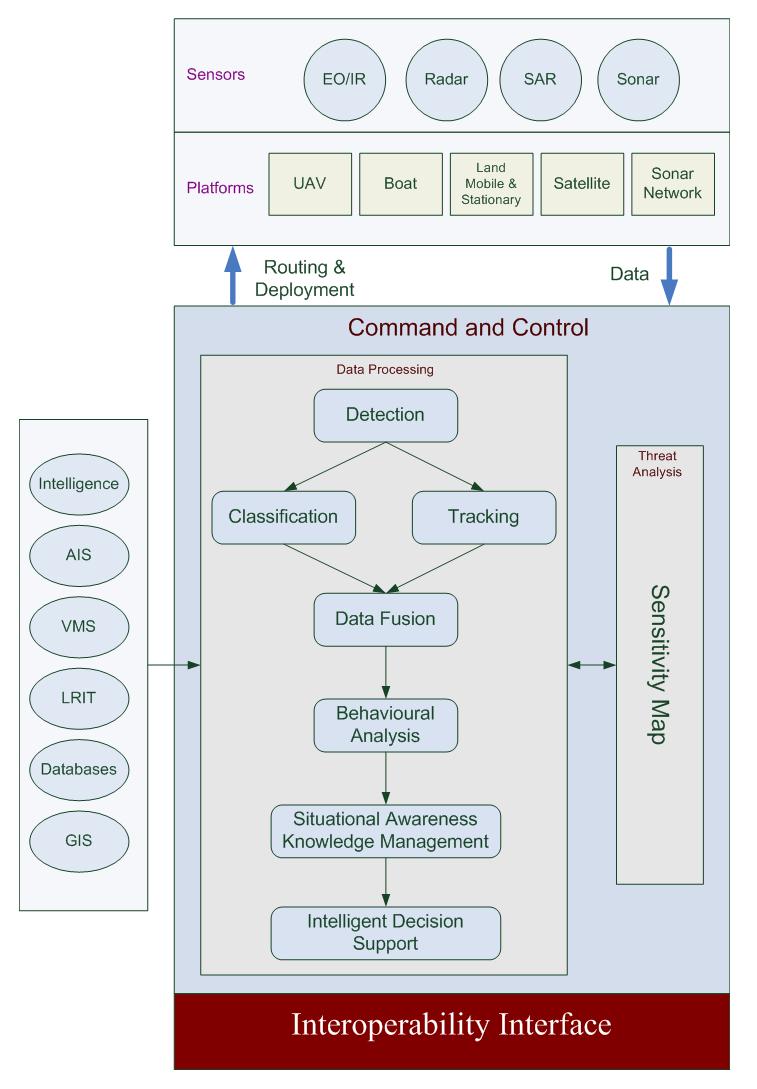


Figure 1 System Architecture

Although there are some maritime surveillance systems available, they lack the technical and architectural maturity to tackle all these requirements at once. Some companies have some of the RECONSURVE subsystems as individual, disparate systems; some have “unified” systems that display several data feeds all at once without the critical automated decision making and support component and yet some have an integrated system with only very limited algorithmic capabilities. A maritime surveillance system with a diverse set of smart sensors installed on various platforms forming a coherent network via interoperability interfaces would address maritime border security needs properly.

The RECONSURVE project proposed as such goes beyond the typical maritime surveillance system with the integration of the following components:

* UAV capabilities
* Sonar network capabilities
* Interoperability
* Automatic detection and classification
* Multi-Sensor Data Analysis

RECONSURVE will address maritime surveillance issues utilizing the system architecture shown in Figure 1. Major work package activities around this architecture will be:

* Requirements Specification
* UAV Software
* Sensor Data Processing
* Developing Situational Awareness
* Interoperability
* Integration
* Validation

The major expected outcomes of the project will be the following:

* Interoperability.
* Small vessel detection & classification capability.
* Cost effectiveness in a wide-area sea border surveillance system.

This deliverable presents the current state of the art in the standards and specifications to be used within the scope of RECONSURVE Project’s interoperability and situational awareness workpackages.

# Introduction

In this deliverable the current state of the art in the standards related with situational awareness and interoperability are presented. In the deliverable first the standards related with situational awareness and then the standards related with interoperability of command and control systems are described.

# Scope

This deliverable presents the current state of the art in the standards and specifications to be used within the scope of RECONSURVE Project.

# Situational Awareness Standards

## Automatic Identification System (AIS)

The Automatic Identification System (AIS) is an automated tracking system used on ships and by Vessel Traffic Services (VTS) for identifying and locating vessels by electronically exchanging data with other nearby ships and VTS stations. AIS information supplements marine radar, which continues to be the primary method of collision avoidance for water transport.

Information provided by AIS equipment, such as unique identification, position, course, and speed, can be displayed on a screen or an ECDIS. AIS is intended to assist a vessel's watch standing officers and allow maritime authorities to track and monitor vessel movements. AIS integrates a standardized VHF transceiver with a positioning system such as a LORAN-C or GPS receiver, with other electronic navigation sensors, such as a gyrocompass or rate of turn indicator. Ships outside AIS radio range can be tracked with the Long Range Identification and Tracking (LRIT) system with less frequent transmission.

The International Maritime Organization's (IMO)[[1]](#footnote-1) International Convention for the Safety of Life at Sea (SOLAS) requires AIS to be fitted aboard international voyaging ships with gross tonnage (GT) of 300 or more tons, and all passenger ships regardless of size. It is estimated that more than 40,000 ships currently carry AIS class A equipment. In 2007, the new Class B AIS standard was introduced which enabled a new generation of low cost AIS transceivers. This has triggered multiple additional national mandates from Singapore, China, Turkey and North America affecting hundreds of thousands of vessels.

### Applications

**Collision avoidance**

AIS is used in navigation primarily for collision avoidance. Due to the limitations of VHF radio communications, and because not all vessels are equipped with AIS, the system is meant to be used primarily as a means of lookout and to determine risk of collision rather than as an automated collision avoidance system, in accordance with the International Regulations for Preventing Collisions at Sea (COLREGS).

When a ship is navigating at sea, the movement and identity of other ships in the vicinity is critical for navigators to make decisions to avoid collision with other ships and dangers (shoal or rocks). Visual observation (unaided, binoculars, night vision), audio exchanges (whistle, horns, VHF radio), and radar or Automatic Radar Plotting Aid (ARPA) are historically used for this purpose. However, a lack of positive identification of the targets on the displays, and time delays and other limitation of radar for observing and calculating the action and response of ships around, especially on busy waters, sometimes prevent possible action in time to avoid collision.

While requirements of AIS are only to display a very basic text information, the data obtained can be integrated with a graphical electronic chart or a radar display, providing consolidated navigational information on a single display.

**Vessel traffic services**

In busy waters and harbors, a local Vessel Traffic Service (VTS) may exist to manage ship traffic. Here, AIS provides additional traffic awareness and provides the service with information on the type of other ships and their movement.

**Aids to navigation**

AIS was developed with the ability to broadcast positions and names of objects other than vessels, like navigational aid and marker positions. These aids can be located on shore, such as in a lighthouse, or on the water or on platforms. The US Coast Guard suggests that AIS might replace RACON, or radar beacons, currently used for electronic navigation aids.

The ability to broadcast navigational aid positions has also created the concepts of Synthetic AIS and Virtual AIS. In the first case, an AIS transmission describes the position of physical marker but the signal itself originates from a transmitter located elsewhere. For example, an on-shore base station might broadcast the position of ten floating channel markers, each of which is too small to contain a transmitter itself. In the second case, it can mean AIS transmissions that indicate a marker which does not exist physically, or a concern which is not visible (i.e., submerged rocks, or a wrecked ship). Although such virtual aids would only be visible to AIS equipped ships, the low cost of maintaining them could lead to their usage when physical markers are unavailable.

**Search and rescue**

For coordinating resources on scene of marine search and rescue operation, it is important to know the position and navigation status of ships in the vicinity of the ship or person in distress. Here AIS can provide additional information and awareness of the resources for on scene operation, even though AIS range is limited to VHF radio range. The AIS standard also envisioned the possible use on SAR Aircraft, and included a message (AIS Message 9) for aircraft to report position.

To aid SAR vessels and aircraft in locating people in distress a standard for an AIS-SART AIS Search and Rescue Transmitter is currently being developed by the International Electrotechnical Commission (IEC), the standard is scheduled to be finished by the end of 2008 and AIS-SARTs will be available on the market from 2009.

**Accident Investigation**

AIS information received by VTS is important for accident investigation to provide the accurate time, identity, position by GPS, compass heading, course over ground (COG), Speed (by log/SOG) and rate of turn (ROT) of the ships involved for accident analysis, rather than limited information (position, COG, SOG) of radar echo by radar.

The maneuvering information of the events of the accident is important to understand the actual movement of the ship before accident, particularly for collision, grounding accidents.

A more complete picture of the events could be obtained by Voyage Data Recorder (VDR) data if available and maintained on board for details of the movement of the ship, voice communication and radar pictures during the accidents. However, VDR data are not maintained due to the limited 12 hours storage by IMO requirement.

### Mechanism

AIS transponders automatically broadcast information, such as their position, speed, and navigational status, at regular intervals via a VHF transmitter built into the transponder. The information originates from the ship's navigational sensors, typically its global navigation satellite system (GNSS) receiver and gyrocompass. Other information, such as the vessel name and VHF call sign, is programmed when installing the equipment and is also transmitted regularly. The signals are received by AIS transponders fitted on other ships or on land based systems, such as VTS systems. The received information can be displayed on a screen or chart plotter, showing the other vessels' positions in much the same manner as a radar display.

The AIS standard comprises several sub-standards 'Types' which specify individual product types. The specification for each product type provides a detailed technical specification which ensures the overall integrity of the global AIS system within which all the product types must operate. The major product types described in the AIS system standards are:

**Class A**

Vessel mounted AIS transceiver (transmit and receive) which operates using self-organized time-division multiple-access (SOTDMA). Class A's must have an integrated display; transmit at 12 W, interface capability with multiple ship systems, and offer a sophisticated selection of features and functions. Default transmit rate is every few seconds. AIS Class A type compliant devices receive all types of AIS messages.

**Class B**

Vessel mounted AIS transceiver (transmit and receive) which operates using carrier-sense time-division multiple-access (CSTDMA). Class Bs transmit at 2 W and are not required to have an integrated display: Class Bs can be connected to most display systems which the received messages will be displayed in lists or overlaid on charts. Default transmit rate is normally every 30 seconds, but this can be varied according to vessel speed or instructions from base stations. The Class B type standard requires integrated GPS and certain LED indicators. Class B equipment receives all types of AIS messages.

**Base station**

Shore based AIS transceiver (transmit and receive) which operates using SOTDMA. Base stations have a complex set of features and functions which in the AIS standard are able to control the AIS system and all devices operating therein.

**Aids to navigation (AtoN)**

Shore or buoy based transceiver (transmit and receive) which operates using fixed-access time-division multiple-access (FATDMA). Designed to collect and transmit data related to sea and weather conditions as well as relay AIS messages to extend network coverage.

AIS receivers are not specified in the AIS standards, because they do not transmit. The main threat to the integrity of any AIS system is non-compliant AIS transmissions, hence careful specifications of all transmitting AIS devices. However, it is well to note that AIS transceivers all transmit on multiple channels as required by the AIS standards. As such single-channel, or multiplexed, receivers will not receive all AIS messages. Only dual-channel receivers will receive all AIS messages.

An AIS transceiver sends the following data every 2 to 10 seconds depending on a vessel's speed while underway, and every 3 minutes while a vessel is at anchor:

* The vessel's [Maritime Mobile Service Identity](http://en.wikipedia.org/wiki/Maritime_Mobile_Service_Identity) (MMSI) – a unique nine digit identification number.
* Navigation status – "at anchor", "under way using engine(s)", "not under command", etc.
* Rate of turn – right or left, from 0 to 720 degrees per minute
* Speed over ground – 0.1-knot (0.19 km/h) resolution from 0 to 102 knots (189 km/h)
* Positional accuracy:
  + Longitude – to 0.0001 minutes
  + Latitude – to 0.0001 minutes
* Course over ground – relative to true north to 0.1°
* True heading – 0 to 359 degrees (for example from a [gyro compass](http://en.wikipedia.org/wiki/Gyro_compass))
* UTC Seconds – The seconds field of the UTC time when these data were generated. A complete timestamp is not present.
* In addition, the following data are broadcast every 6 minutes:
* [IMO ship identification number](http://en.wikipedia.org/wiki/IMO_ship_identification_number) – a seven digit number that remains unchanged upon transfer of the ship's registration to another country
* [Radio call sign](http://en.wikipedia.org/wiki/Call_sign#Ships_and_boats) – international radio call sign, up to seven characters, assigned to the vessel by its country of registry
* Name – 20 characters to represent the name of the vessel
* Type of ship/cargo
* Dimensions of ship – to nearest meter
* Location of positioning system's (e.g., GPS) antenna on board the vessel - in meters aft of bow and meters port of starboard
* Type of positioning system – such as [GPS](http://en.wikipedia.org/wiki/Global_Positioning_System), [DGPS](http://en.wikipedia.org/wiki/Differential_GPS) or [LORAN-C](http://en.wikipedia.org/wiki/LORAN).
* [Draught](http://en.wikipedia.org/wiki/Draft_(hull)) of ship – 0.1 meter to 25.5 meters
* Destination – max. 20 characters
* ETA (estimated time of arrival) at destination – UTC month/date hour:minute

## OGC Sensor Web Enablement (SWE)

This section describes the architecture implemented by Open Geospatial Consortium’s (OGC) [[2]](#footnote-2)Sensor Web Enablement Initiative (SWE)[[3]](#footnote-3). In much the same way that HTML and HTTP standards enabled the exchange of any type of information on the Web, the SWE initiative is focused on developing standards to enable the discovery of sensors and corresponding observations, exchange, and processing of sensor observations, as well as the tasking of sensors and sensor systems. The functionality that OGC has targeted within the Sensor Web includes:

* Discovery of sensor systems, observations, and observation processes that meet our immediate needs
* Determination of a sensor’s capabilities and quality of measurements
* Access to sensor parameters that automatically allow software to process and geo-locate observations
* Retrieval of real-time or time-series observations and coverages in standard encodings
* Tasking of sensors to acquire observations of interest
* Subscription to and publishing of alerts to be issued by sensors or sensor services based upon certain criteria

The Sensor Web represents a meta-platform that integrates arbitrary sensors and sensor networks; each maintained and operated by individual institutions, e.g. the Australian Water Resources Network, the European Environment Information and Observation Network, or the South African Earth Observation Network. This reflects the existing legal, organizational and technical situation. Sensors and sensor systems are operated by various organizations with varying access constraints, security, and data quality and performance requirements. The architectural design of the Sensor Web allows the integration of individual sensors as much as the integration of complete sensor systems without the need of fundamental changes to the legacy systems.

Within the SWE initiative, the enablement of such sensor webs is being pursued through the establishment of the following encodings for describing sensors and sensor observations,

1. Sensor Model Language (SensorML) – standard models and XML Schema for de-scribing sensors systems and processes; provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing taskable properties
2. Observations and Measurements Schema (O&M) – standard models and XML Schema for encoding observations and measurements from a sensor, both archived and real-time and through four standard interface definitions for Web services:
3. Sensor Observations Service (SOS) – standard web service interface for requesting, filtering, and retrieving observations and sensor system information
4. Sensor Planning Service (SPS) – standard web service interface for requesting user-driven acquisitions and observations, to (re-)calibrate a sensor or to task a sensor network
5. Sensor Alert Service (SAS) – standard web service interface for publishing and subscribing to alerts from sensors
6. Web Notification Services (WNS) – standard web service interface for asynchronous delivery of messages or alerts from any other web service

### Sensor Markup Language (SensorML)

The measurement of phenomena that results in an observation consists of a series of processes, beginning with the processes of sampling and detecting and followed perhaps by processes of data manipulation. The division between measurement and “post-processing” has become blurred with the introduction of more complex and intelligent sensors, as well as the application of more on-board processing of observations. The typical Global Positioning System (GPS) sensor is a prime example of a device that consists of basic detectors complemented by a series of complex processes that result in the observations of position, heading, and velocity. SensorML defines models and XML Schema for describing any process, including measurement by a sensor system, as well as post-measurement processing. SensorML supports a variety of needs within the sensor community, including:

**Discovery of sensor, sensor systems, and processes** - SensorML is a means by which sensor systems or processes can make themselves known and discoverable. SensorML provides a rich collection of metadata that can be mined and used for discovery of sensor systems and observation processes.

**On-demand processing of Observations** - Process chains for geolocation or higher-level processing of observations can be described in SensorML, discovered and distributed over the web, and executed on-demand without a priori knowledge of the sensor or processor characteristics.

**Lineage of Observations** - SensorML can provide a complete and unambiguous description of the lineage of an observation. In other words, it can describe in detail the process by which an observation came to be, covering the entire life span from acquisition by one or more detectors to processing and perhaps even interpretation by an analyst. Not only can this provide a confidence level with regard to an observation, in most cases, part or all of the process could be repeated, perhaps with some modifications to the process or by simulating the observation with a known signature source.

**Support for tasking, observation, and alert services** - SensorML descriptions of sensor systems or simulations can be mined in support of establishing OGC Sensor Observation Services (SOS), Sensor Planning Services (SPS), and Sensor Alert Services (SAS). SensorML defines and builds on common data definitions that are used throughout the OGC Sensor Web Enablement (SWE) framework.

**Plug-N-Play, auto-configuring, and autonomous sensor networks** - SensorML enables the development of plug-n-play sensors, simulations, and processes, which seamlessly be added to Decision Support systems. The self-describing characteristic of SensorML-enabled sensors and processes also supports the development of auto-configuring sensor networks, as well as the development of autonomous sensor networks in which sensors can publish alerts and tasks to which other sensors can subscribe and react.

**Archiving of Sensor Parameters** - Finally, SensorML provides a mechanism for archiving fundamental parameters and assumptions regarding sensors and processes, so that observations from these systems can still be reprocessed and improved long after the origin mission has ended. This is proving to be critical for long-range applications such as global change monitoring and modeling.

In , a sample SensorML definition for a sensor making air temperature, atmospheric pressure, relative humidity and visibility observations is given.



Figure 2 Sample SensorML description

### Observations and Measurements (O&M)

Observations & Measurements (OM) provides general models and schema for supporting the packaging of observations from sensor system and sensor-related processing. The model supports metadata about the Observation, as well as the ability to link to the procedure (i.e. sensors plus processing) that created the observation, thus, providing an indication of the lineage of the measurements. Through the O&M specification, the SWE framework provides a standard XML-based package for returning observation results. In , as sample observation is presented.

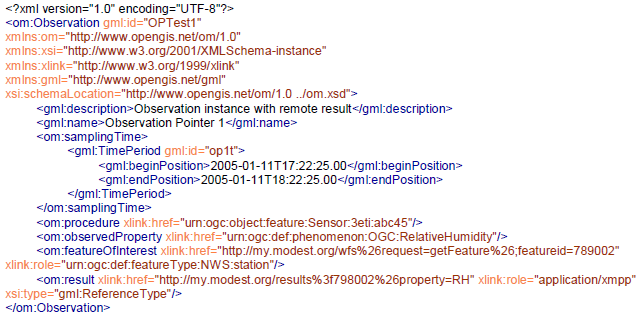


Figure 3 Sample Observation

### Sensor Observation Service (SOS)

The OpenGIS Sensor Observation Service Interface Standard (SOS) provides an API for managing deployed sensors and retrieving sensor data and specifically “observation” data. Whether from in-situ sensors (e.g., water monitoring) or dynamic sensors (e.g., satellite imaging), measurements made from sensor systems contribute most of the geospatial data by volume used in geospatial systems today. An SOS organizes collections of related sensor system observations into Observation Offerings. An Observation Offering is also analogous to a “layer” in Web Map Service because each offering is intended to be a non-overlapping group of related observations. Each Observation Offering is constrained by a number of parameters including the following:

* Specific sensor systems that report the observations,
* Time period(s) for which observations may be requested (supports historical data),
* Phenomena that are being sensed,
* Geographical region that contains the sensors, and
* Geographical region that is the subject of the sensor observations (may differ from the sensor region for remote sensors)

SOS carefully models sensors, sensor systems, and observations in such a way that the model covers all varieties of sensors and supports the requirements of all users of sensor data. In , the interaction diagram on how a consumer can retrieve sensor data from SOS’ is displayed.

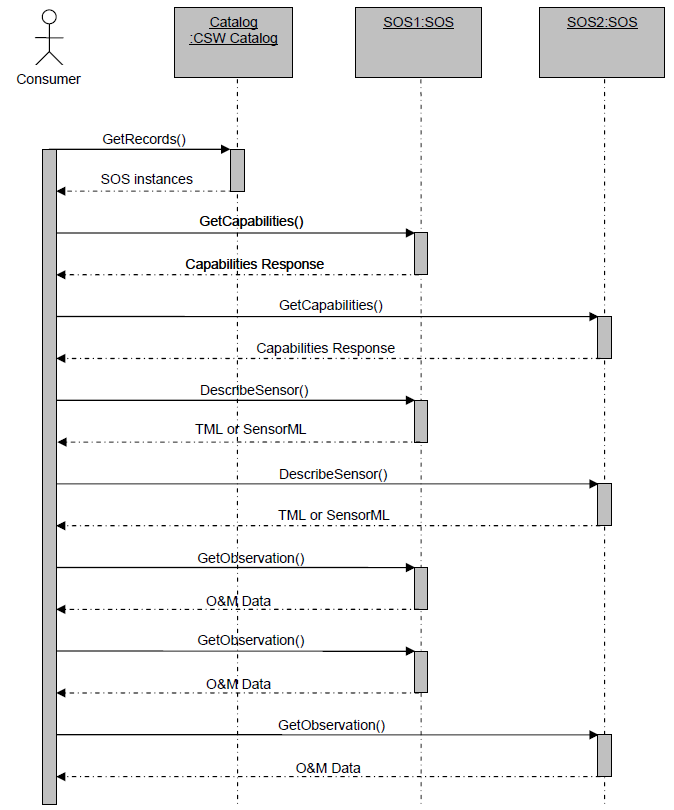


Figure 4 Sensor Data Consumer Sequence Diagram

### Sensor Planning Service (SPS)

The Sensor Planning Service (SPS) is intended to provide a standard interface to collection assets (i.e., sensors, and other information gathering assets) and to the support systems that surround them. The OpenGIS® Sensor Planning Service Interface Standard (SPS) defines interfaces for queries that provide information about the capabilities of a sensor and how to task the sensor. The standard is designed to support queries that have the following purposes:

* to determine the feasibility of a sensor planning request;
* to submit such a request;
* to inquire about the status of such a request;
* to update or cancel such a request; and
* to request information about other OGC Web services that provide access to the data collected by the requested task.

The interaction sequence basically consists of the following steps as shown in :

* Discovery of SPS as a service that is capable of providing required data sets (given that the required data was not provided by an SOS already).
* Requesting the list of parameters that have to be set in order to start the SPS process (the backend of an SPS is opaque to the user, but described using SensorML, i.e. the user is provided with all information about the process itself, but gets no information about the communication between SPS and the process)
* Providing values for all required parameters and execution of the process
* Receiving information about the availability of the produced data at an OGC interface, e.g. SOS

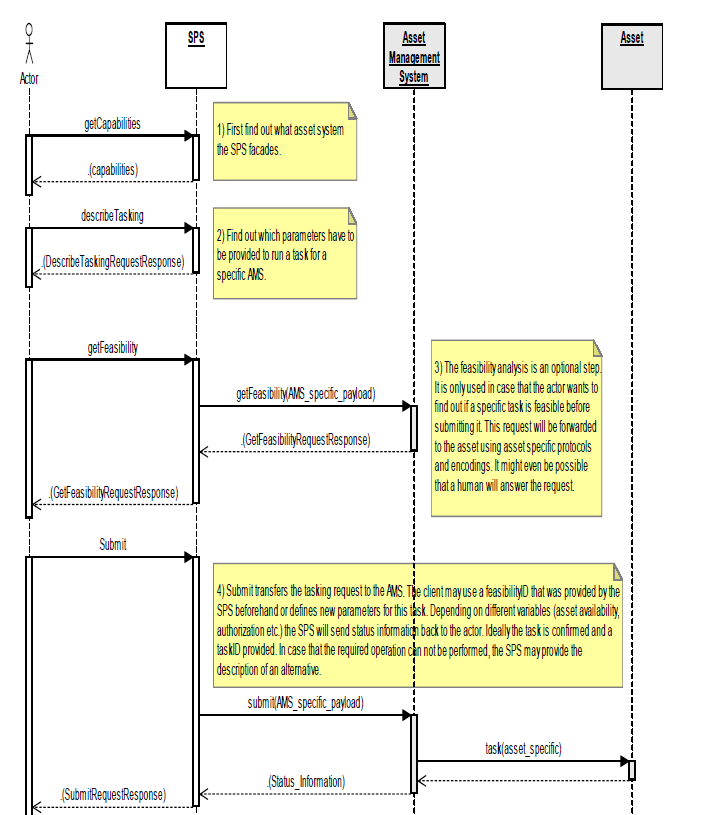


Figure 5 SPS Interaction Diagram

### Sensor Alert Service (SAS)

The Sensor Alert Service (SAS) can be compared with an event stream processor in combination with an event notification system. An SAS processes incoming sensor data continuously. Pattern-matching algorithms identify satisfied alert conditions and start the alert distribution process. An SAS might therefore provide a wide variety of alerts related to sensors and sensor observations including, as examples, measured values above a threshold, detected motion or the presence of a recognizable feature, or perhaps sensor status (e.g. low battery, shut-down or startup). An SAS can advertise what alerts it can provide. A consumer (interested party) may subscribe to alerts disseminated by the SAS. If an event occurs the SAS will publish an alert and notify all clients subscribed to this event type through a messaging service. Currently, SAS supports XMPP (Extensible Messaging and Presence Protocol) for alert distribution exclusively, although in combination with a Web Notification Service (WNS), it may deliver alerts to other communication endpoints as well. The sequence diagram is shown in . As shown in the figure, the sensor first advertises itself to the SAS. After that it publishes the observed values to SAS through XMPP protocol. A corresponding subscribed client will be notified about this (if subscription criteria holds) value through either XMPP protocol or Web Notification Service.

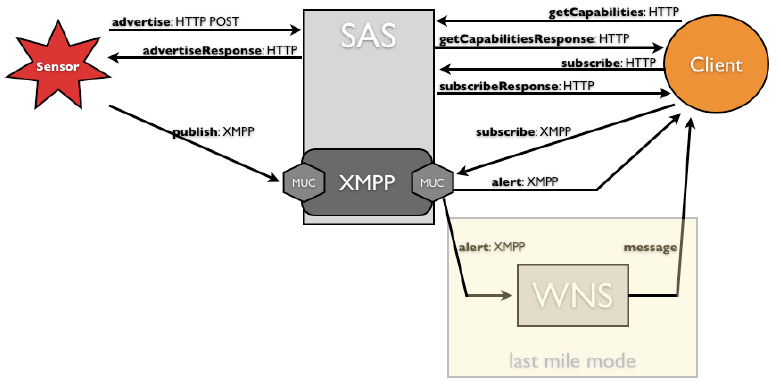


Figure 6 SAS Communication Diagram

### Web Notification Service (WNS)

The Web Notification Service Model includes two different kinds of notifications. First, the “one-way-communication” provides the user with information without expecting a response. Second, the “two-way-communication” provides the user with information and expects some kind of asynchronous response.

The basis on which notifications will be sent is free to the service and will be described in its capabilities. The “way-of-notification” palette may include:

* e-mail
* http-call (as HTTP POST: in case of sophisticated clients that act as web services themselves)
* SMS
* Instant Message
* phone call
* etter
* fax
* XMPP
* any other communication protocol

Once a client registers a user along with the method of notification desired, the client receives a unique RegistrationID that can then be provided as input to other services (e.g. SPS or SAS). In , a sample registration request and notification message is displayed.

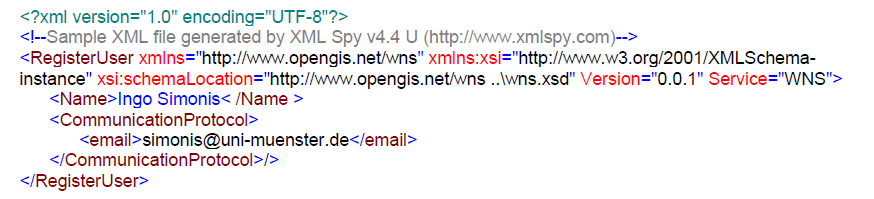


Figure 7 WNS registration and notification sample

### SWE Services Usages

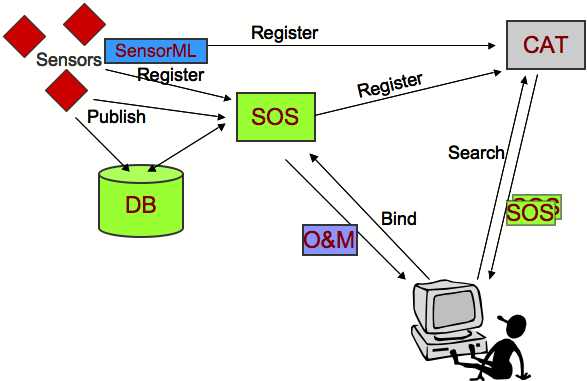


Figure 8 SOS and O&M usage

shows sensors that are registered at a SOS and publish observation results to the services itself (then we speak of a Transactional-SOS) or into a database which will be accessed by the SOS service instance. To be discoverable, sensors, using SensorML, and SOS register at a catalog service. The user in the lower left corner requires observation data and sends therefore a search request to the catalog. The catalog responses with a list of SOS service instances that fulfill the requirements. Eventually, the user binds the SOS and retrieves the observation data, encoded in O&M.

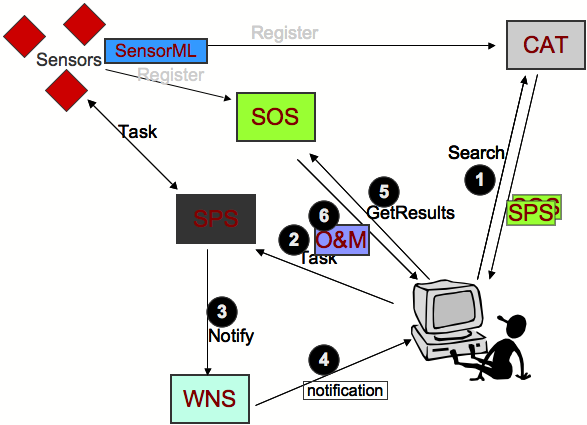


Figure 9 SOS, SPS, WNS and O&M usage

The situation gets a bit more complex in . Let’s assume that the catalog didn’t provide any SOS instances that fulfill the requirements set by the user. In this case, the user may search for Sensor Planning Services that could task sensors to perform appropriate actions in order to produce the observation data our user is looking for. The catalog provides the link to the SPS instance and the user assigns the task. The SPS forwards the command to the sensor. The communication between the SPS and the sensor is opaque for the user. If we imagine a satellite that has to reorient its infrared cameras and has to reach its target position in space, the tasking might take a while. Once observing the requested scene, the sensor dumps its data into a database that is linked to a SOS. The SPS informs the user about data availability using a Web Notification Service. This has the advantage that the SPS can respond to the tasking request right away and has a mechanism to reach the user at a later stage, e.g. if the data is available or if the tasking is delayed or cancelled. The notification message contains all necessary information to access the data from a SOS.

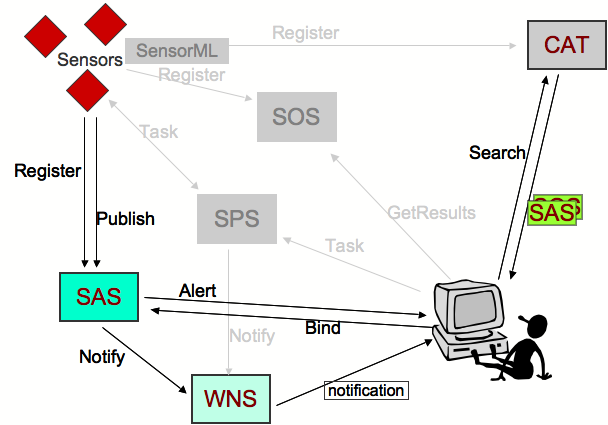


Figure 10 SAS and WNS Usage

Another scenario is shown in , the client receives information about appropriate SAS from a catalog and subscribes to the SAS. Sensors publish observation results continuously to the SAS. The SAS handles all the filtering and alerts the client if the subscription condition is matched. The SAS either sends the alert directly to the client, or makes use of the WNS in order to deliver the alert message.

## Description Logics Reasoners

Currently, there are the following Description Logics reasoners in the literature: Racer Pro[[4]](#footnote-4), KAON2[[5]](#footnote-5), Fact++[[6]](#footnote-6) and Pellet[[7]](#footnote-7). A recent survey[[8]](#footnote-8) investigates the resoners considering their OWL support, correctness, efficiency, interface capabilities and inference services. The survey concludes that no system, except RacerPro and KAON2, is able to correctly solve at least those tests which lay within the language fragment that the tools claim to support in full. And to some extend KAON2 is not application ready since it fails very often with “out of memory errors” or requires significant processing time for language constructs, which are typically in real-world models such as cardinality restrictions. Pellet and FaCT++ do have some serious bugs which result in incorrect answers. In addition to the survey, in the scope of the thesis, the above mentioned reasoners are investigated in terms of their efficiency. Only Racer Pro could answer to the situational awareness ontology without “out of memory error”. Therefore, in this project Racer Pro is used as the Description Logics Reasoner and this reasoner will be used to develop/maintain the Situational Awareness Ontology.

### RULE ENGINES AND RULE LANGUAGES

#### JESS

Jess[[9]](#footnote-9) is a rule engine and scripting environment written entirely in Sun's Java language. Using Jess, one can build Java software that has the capacity to "reason" using knowledge supplied in the form of declarative rules. Jess is small, light, and one of the fastest rule engines available. Its powerful scripting language gives access to all of Java's APIs. Jess includes a full-featured development environment based on the Eclipse platform.

Jess uses an enhanced version of the Rete algorithm to process rules. Rete[[10]](#footnote-10) is a very efficient mechanism for solving the difficult many-to-many matching problem. Jess has many features including backwards chaining and working memory queries, and Jess can directly manipulate and reason about Java objects. Jess is also a powerful Java scripting environment, from which one can create Java objects, call Java methods, and implement Java interfaces without compiling any Java code.

Jess can be licensed for commercial use, and is available at no cost for academic use.

#### DROOLS

Drools[[11]](#footnote-11) is a business rule management system (BRMS) with a forward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm. Drools supports the JSR-94 standard for its business rule engine and enterprise framework for the construction, maintenance, and enforcement of business policies in an organization, application, or service. The rule engine and rule format of Drools is currently supported by well-known open-source business process engines such as jBPM[[12]](#footnote-12) and Activiti[[13]](#footnote-13).

Drools is designed to allow pluggable language implementations. Currently, rules can be written in Java, MVEL, Python, and Groovy. Drools also provides for declarative programming and is flexible enough to match the semantics of the problem domain with domain specific languages (DSL) via XML using a schema defined for the problem domain. DSLs consist of XML elements and attributes that represent the problem domain.

#### Semantic Web Rule Language (SWRL)

SWRL (Semantic Web Rule Language)[[14]](#footnote-14) is a proposal for a Semantic Web rules-language, combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the RuleML[[15]](#footnote-15) - Rule Markup Language (Unary/Binary Datalog). The specification was submitted in May 2004 to the W3C by the National Research Council of Canada, Network Inference and Stanford University in association with the Joint US/EU ad hoc Agent Markup Language Committee. SWRL includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

As an example, assume that there is such a rule among the classes of an ontology:

hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)

The actual SWRL syntax for this rule is a combination of the OWL Web Ontology Language XML Presentation Syntax with the RuleML XML syntax:

**<ruleml:imp>**

**<ruleml:\_rlab** ruleml:href="#example1"**/>**

**<ruleml:\_body>**

**<swrlx:individualPropertyAtom** swrlx:property="hasParent"**>**

**<ruleml:var>**x1**</ruleml:var>**

**<ruleml:var>**x2**</ruleml:var>**

**</swrlx:individualPropertyAtom>**

**<swrlx:individualPropertyAtom** swrlx:property="hasBrother"**>**

**<ruleml:var>**x2**</ruleml:var>**

**<ruleml:var>**x3**</ruleml:var>**

**</swrlx:individualPropertyAtom>**

**</ruleml:\_body>**

**<ruleml:\_head>**

**<swrlx:individualPropertyAtom** swrlx:property="hasUncle"**>**

**<ruleml:var>**x1**</ruleml:var>**

**<ruleml:var>**x3**</ruleml:var>**

**</swrlx:individualPropertyAtom>**

**</ruleml:\_head>**

**</ruleml:imp>**

Currently there is no specific rule engine that supports SWRL rules. The well accepted common practice to execute these rules is first to convert them into executable rule formats such as Jess and then execute them in the corresponding rule engine. In the scope of RECONSURVE Project, we will apply a similar approach. The situational awareness rules will be defined with SWRL syntax and in terms of situational awareness ontology classes and they will be executed with the help of Jess or Drools rule engines.

# Interoperability Standards

## Joint Consultation, Command and Control Information Exchange Data Model (JC3IEDM)

Unilateral capability is important to nations but most planning is made on the assumption of alliance and coalition operations in scenarios that are difficult to predict and which often arise at short notice. Thus the nature and composition of a force structure to meet military requirements will be specific to requirement and based upon a general and flexible military capability. To achieve this, an assured capability for interoperability of information is essential. The successful execution of fast moving operations needs an accelerated decision-action cycle, increased tempo of operations, and the ability to conduct operations within combined joint formations. Commanders require timely and accurate information. Also, supporting command and control (C2) systems need to pass information within and across national and language boundaries. Moreover, tactical C2 information must be provided to the operational and strategic levels of command including other governmental departments. Additionally, forces must interact with non-governmental organizations, including international aid organizations. In this respect, the Multilateral Interoperability Programme (MIP)[[16]](#footnote-16) aims to deliver an assured capability for interoperability of information to support joint / combined operations.

The aim of the Multilateral Interoperability Programme (MIP) is to achieve international interoperability of Command and Control Information Systems (C2IS) at all levels from corps to battalion, or lowest appropriate level, in order to support multinational (including NATO), combined and joint operations and the advancement of digitization in the international arena.

Towards this aim, MIP produced Joint C3 Information Exchange Data Model (JC3IEDM)[[17]](#footnote-17) which is a model that when physically implemented aims to enable the interoperability of systems and projects required to share Command and Control (C2) information. JC3IEDM is an evolution of the C2IEDM[[18]](#footnote-18) standard that includes joint operational concepts, just as the Land Command and Control Information Exchange Data Model (LC2IEDM) was extended to become C2IEDM.

### Overview of the Data Model

JC3IEDM is intended to represent the core of the data identified for exchange across multiple functional areas and multiple views of the requirements. Toward that end, it lays down a common approach to describing the information to be exchanged in a command and control (C2) environment.

a. The structure should be sufficiently generic to accommodate joint, land, sea, and air environmental concerns.

b. The data model describes all objects of interest in the sphere of operations, e.g., organisations, persons, equipment, facilities, geographic features, weather phenomena, and military control measures such as boundaries.

c. Objects of interest may be generic in terms of a class or a **type** and specific in terms of an individually identified **item**. All object *items* must be classified as being of some *type* (e.g. a specific tank that is identified by serial number WS62105B is an item of type "Challenger" that is a heavy UK main battle tank).

d. An object must have the capability to perform a function or to achieve an end. Thus, a description of capability is needed to give meaning to the value of objects in the sphere of operations.

e. It should be possible to assign a location to any item in the sphere of operations. In addition, various geometric shapes need to be represented in order to allow commanders to plan, direct, and monitor operations. Examples include boundaries, corridors, restricted areas, minefields, and any other control measures needed by commanders and their staffs.

f. Several aspects of status of items need to be maintained.

g. The model must permit a description of the composition of a type object in terms of other type objects. Such concepts include tables of organisations, equipment, or personnel.

h. The model must reflect information about what is held, owned or possessed in terms of types by a specific object item.

i. There is a need to record relationships between pairs of items. Key among these is the specification of unit task organisations and orders of battle.

j. The model must support the specification of current, past, and future role of objects as part of plans, orders, and events.

k. The same data structure should be used to record information for all objects, regardless of their hostility status.

l. Provision must be made for the identification of sources of information, the effective and reporting times, and an indication of the validity of the data.

The overall goal is to specify the minimum set of data that needs to be exchanged in coalition or multinational operations. Each nation, agency or community of interest is free to expand its own data dictionary to accommodate its additional information exchange requirements with the understanding that the added specifications will be valid only for the participating nation, agency or community of interest.

#### Foundational Structural Elements

##### Entities

Basic concept in data specification is an entity, i.e., any distinguishable person, place, thing, event, or concept about which information is to be kept. Properties or characteristics of an entity are referred to as attributes. The attributes make explicit the data that is to be recorded for each concept of interest. This edition of the model contains 271 entities. The entire structure is generated from 19 independent entities, that is, entities whose identification does not depend on any other entity. All other entities are dependent entities. Independent entities are defined in . The general role that each entity serves is also suggested.

Table 1. Independent Entities and Their Roles

| Entity Name | Entity Definition | **Role in the Model** |
| --- | --- | --- |
| ACTION | An activity, or the occurrence of an activity, that may utilise resources and may be focused against an objective. Examples are operation order, operation plan, movement order, movement plan, fire order, fire plan, fire mission, close air support mission, logistics request, event (e.g., incoming unknown aircraft), or incident (e.g., enemy attack). | Dynamics  (How, what, when something is to be done, is being done, or has been done.) |
| ADDRESS | Precise information on the basis of which a physical or electronic destination may be accessed. | Provides means to record postal and electronic addresses. |
| AFFILIATION | A specification of a country, nationality, ethnic group, functional group, exercise group, or religion to which membership or allegiance may be ascribed. | Provides means to assign affiliations to type or item objects. |
| CANDIDATE-TARGET-LIST | A list of selected battlespace objects or types that have potential value for destruction or exploitation, nominated by competent authority for consideration in planning battlespace activities. | Information to support ACTION. |
| CAPABILITY | The potential ability to do work, perform a function or mission, achieve an objective, or provide a service. | Indication of expected capability for types and actual capability for items |
| COMPONENT-HEADER-CONTENT | Introductory subject matter intended to identify an element of a plan or order. | Used in conjunction with plan and order specifications. |
| COMPONENT-TEXT-CONTENT | A textual statement of substantive subject matter. | Used in conjunction with plan and order specifications. |
| CONTEXT | A collection of information that provides in its entirety the circumstances, conditions, environment, or perspective for a situation. | Multiple roles including grouping of information. |
| RELATIVE-COORDINATE-SYSTEM | A rectangular frame of reference defined by an origin, x and y axes in the horizontal plane, and a z-axis. The vertical z-axis is normal to the xy-plane with positive direction determined from the right-hand rule when the x-axis is rotated toward the y-axis. | Support to LOCATION for specifying relative geometry. |
| GROUP-CHARACTERISTIC | A reference to a set of characteristics that may be used for identifying a distinct collection of objects. Examples of characteristics include age group, malady, gender, language, and triage classification. | Supports the counting of types of persons according to selected characteristics. |
| LOCATION | A specification of position and geometry with respect to a specified horizontal frame of reference and a vertical distance measured from a specified datum. Examples are point, sequence of points, polygonal line, circle, rectangle, ellipse, fan area, polygonal area, sphere, block of space, and cone. LOCATION specifies both location and dimensionality. | Geopositioning of objects and creation of shapes  (Where) |
| OBJECT-ITEM | An individually identified object that has military or civilian significance. Examples are a specific person, a specific item of materiel, a specific geographic feature, a specific coordination measure, or a specific unit. | Identifying individual things.  (Who and What) |
| OBJECT-TYPE | An individually identified class of objects that has military or civilian significance. Examples are a type of person (e.g., by rank), a type of materiel (e.g., self-propelled howitzer), a type of facility (e.g., airfield), a type of feature (e.g., restricted fire area), or a type of organisation (e.g., armoured division). | Identifying classes of things.  (Who and What) |
| PLAN-ORDER | A planned or ordered scheme worked out beforehand for the accomplishment of an operational objective. | The top-level entity for identification of a plan or order. |
| REFERENCE | Identification of a record of information. | Pointing to external information in support of REPORTING-DATA. |
| REPORTING-DATA | The specification of source, quality and timing that applies to reported data. | Support for the reporting function. |
| RULE-OF-ENGAGEMENT | A specification of mandatory guidance for the way a given activity is to be executed. | Support to ACTION. |
| SECURITY-CLASSIFICATION | The security classification applicable to an information resource within the domain of classified security information. | Support to CONTEXT, NETWORK-SERVICE and REFERENCE |
| VERTICAL-DISTANCE | A specification of the altitude or height of a point or a level as measured with respect to a specified reference datum in the direction normal to the plane that is tangent to the WGS84 ellipsoid of revolution. | Support to LOCATION in specifying elevation or height. |

The independent entities and their relationships are illustrated in . A dot at the end of a relationship line denotes “many.” The relationships shown in this diagram are either many-to-many (solid line with two dots) or non-identifying one-to-many (dashed line). For example, the relationship between OBJECT-ITEM and LOCATION is to be interpreted as a pair of statements that an OBJECT-ITEM may have zero, one, or more LOCATIONs and that a LOCATION may apply to zero, one, or more OBJECT-ITEMs. The entities that connect to the rest of the structure by means of non-identifying relationships provide auxiliary specifications that are needed for precise definition of the concepts that are being captured. Some of the relationships are recursive, such as those relating ACTION to itself.



Figure 11. Independent Entities for Creating the Data Specification

“Things” must be identified as the first step—who are the actors and what things are available to be used by or are used by the actors? Model design encompasses two categories of objects: those that can be identified individually (by name—2 (SP) Armoured Cavalry Brigade, Jubilation T. Cornpone, by call sign or serial number or license plate or passport number, and so on) and those that represent grouped or class properties (a tank, a ship, an M1A2 tank, a helicopter, a howitzer, a rifle, an armoured brigade, a light infantry battalion, an infantryman, a refugee). The two categories are used in parallel as basic structural elements of the model. The two structures are related to each other. Data characteristics are entered either on the item side or the type side as appropriate. Any characteristic described on the type side also applies to the item when the item is assigned a type classification. The linkage from item to type is mandatory in the model.

JC3IEDM uses the name OBJECT-TYPE to refer to class objects and OBJECT-ITEM for individually identified instances. Implicit in the distinction between type and item is the assumption that data relating to OBJECT-TYPEs will tend to be relatively static or persistent (i.e., the values of the attributes are not likely to change very often over time), whereas the data characteristics related to OBJECT-ITEMs are likely to be more dynamic. For example, if a characteristic is about a type (e.g., M1A1 Abrams Tank), it is an attribute of OBJECT-TYPE. Thus, calibre of main gun, track width, and load class are characteristics of OBJECT-TYPE. However, the call sign, actual fuel level, munitions holdings, and current operational status of a specific tank are characteristics of an OBJECT-ITEM. At the same time, the mandatory classification of an instance of OBJECT-ITEM as an instance of OBJECT-TYPE assures that the item inherits all the characteristics of the type.

Item and type objects are subdivided into extensive hierarchies. The first-level hierarchy is parallel and is illustrated in . There are five categories or subtypes to encompass any object within the scope of the model: facility, feature, materiel, organization, and person. A subtype is the same thing as its parent, but it has some properties that do not apply to its siblings. Complete subtyping is denoted by a double line under the circle. It means that no other category is needed in response to the set of requirements that governed evolution of the model. A circle with one line underneath is a symbol for incomplete subtyping. It means that more subtypes could be defined if needed. Definitions of subtype entities are presented in . As may be expected, the two sets of definitions are similar.



Figure 12. First Level Subtyping of OBJECT-TYPE and OBJECT-ITEM

Table 2. Definition of First-Level Subtypes

|  |  |
| --- | --- |
| Entity | Entity Definition |
| FACILITY | An OBJECT-ITEM that is built, installed, or established to serve some particular purpose and is identified by the service it provides rather than by its content. |
| FACILITY-TYPE | An OBJECT-TYPE that is intended to be built, installed or established to serve some particular purpose and is identified by the service it is intended to provide rather than by its content. Examples include a refuelling point, a field hospital, and a command post. |
| FEATURE | An OBJECT-ITEM that encompasses meteorological, geographic, and control features of military significance. |
| FEATURE-TYPE | An OBJECT-TYPE that encompasses meteorological, geographic, and control features of military significance. Examples include a forest, an area of rain, a river, an area of responsibility. |
| MATERIEL | An OBJECT-ITEM that is equipment, apparatus or supplies of military interest without distinction as to its application for administrative or combat purposes. |
| MATERIEL-TYPE | An OBJECT-TYPE that represents equipment, apparatus or supplies of military interest without distinction to its application for administrative or combat purposes. Examples include ships, tanks, self-propelled weapons, aircraft, etc., and related spares, repair parts, and support equipment, but excluding real property, installations, and utilities. |
| ORGANISATION | An OBJECT-ITEM that is an administrative or functional structure. |
| ORGANISATION-TYPE | An OBJECT-TYPE that represents administrative or functional structures. |
| PERSON | An OBJECT-ITEM that is a human being to whom military or civilian significance is attached. |
| PERSON-TYPE | An OBJECT-TYPE that represents human beings about whom information is to be held. |

##### OBJECT-TYPE HIERARCHY

The OBJECT-TYPE subtyping tree is extended beyond the first level as illustrated in . FEATURE-TYPE has two subtypes, FACILITY-TYPE has four subtypes, MATERIEL-TYPE and ORGANISATION-TYPE have extensive subtype hierarchies; and PERSON-TYPE has no subtypes. Categorization of OBJECT-TYPE can be done in different ways. There is no right or wrong way. The structure described in the figure happens to satisfy the stated information exchange requirements most closely.



Figure 13. Entity-Level View of OBJECT-TYPE Subtype Tree

Most of the categories are reasonably self-explanatory with the possible exception of GROUP-ORGANISATION-TYPE, CIVILIAN-POST-TYPE, and MILITARY-POST-TYPE. GROUP-ORGANISATION-TYPE was created in response to CRO requirements to deal with groups that are not truly organizations but have to be treated as a collective object for data purposes. Consequently, groups of people such as refugees and prisoners of war are treated as pseudo-organizations. Post type is a type of position that is filled by a single individual, such as commander of a military unit or chief of a police department. It enables the set of duties inherent in a position or a billet to be distinguished from the type of person that may fill the post.

##### OBJECT-ITEM Hierarchy

The independent entity OBJECT-ITEM identifies individual instances of objects and records information concerning individual characteristics that are not typical of the class. OBJECT-ITEM is broken down into a hierarchy of subtypes. Full OBJECT-ITEM subtype hierarchy is illustrated in .



Figure 14. Entity-Level View of OBJECT-ITEM Subtype Tree

##### Plans and Orders

The basic operational requirements for plans and orders are the provisions of STANAG 2014 Edition 9. The data schema is designed to:

a. Satisfy most STANAG 2014 requirements in storing a plan or order in data and maintaining the proper structure or paragraphing,

b. Enable the use of the ACTION and other JC3IEDM specifications of structured data to represent those parts of a plan or order that are appropriate for structured data,

c. Permit the use of textual information to specify those parts of a plan or order that cannot be expressed as structured data,

d. Permit the use of textual information to supplement those parts of a plan or order that are represented as structured data.

The structure is shown in .

PLAN-ORDER is the top-level entity through which warning orders, plans, operation orders, fragmentary orders, separate annexes, and any other document identified in STANAG 2014 can be managed. The content that applies to the entirety of a PLAN-ORDER is specified in PLAN-ORDER-HEADER-CONTENT.

The detailed content for an instance of PLAN-ORDER is specified using the child entity PLAN-ORDER-COMPONENT. It serves as the basis for collecting all of the information that is attendant to the component.



Figure 15. Plans and Orders Structure Shown at Entity Level

##### ACTION: Structured Specification of Activity

The independent entity ACTION is the root for representation of the all types of activities. The related structure includes mechanisms for specifying items or types as resources and objectives for activity, recording effects of activity, classifying activities as planned tasks or unplanned events, keeping status of activities, and relating activities to each other functionally and temporally.

ACTION together with its substructures specifies and describes operations planned for, or carried out, in the sphere of operations. It is also used to describe unplanned happenings that are of military interest. The underlying concept for modeling ACTIONs is based on a statement in which something carries out an activity to affect something at some time. Within the model, the "something" within the basic action statement is described by an OBJECT-TYPE or an OBJECT-ITEM. Thus, OBJECT-TYPEs and OBJECT-ITEMs are related to ACTION in three distinct ways: as resources objectives, and subjects of effects, as illustrated in . The figure also shows two associations that link sets of ACTIONs functionally and temporally. Functional associations enable the building of complex statements, such as operations orders, from simple statements in cascading hierarchies. Temporal associations provide for timing of ACTIONs in relation to one another in specific or general terms.

ACTION-RESOURCE is defined as “An OBJECT-ITEM or an OBJECT-TYPE that is required, requested, allocated or otherwise used or planned to be used in conducting a specific ACTION.” ACTION-RESOURCEs are those OBJECT-ITEMs and OBJECT-TYPEs that have been specified as the things performing, things being used in or allocated to, or things whose use is qualified in some way, in carrying out a specific ACTION.

ACTION-OBJECTIVE is defined as “The focus, in terms of an OBJECT-ITEM, OBJECT-TYPE, or ACTION-TASK, in conducting a specific ACTION.” ACTION-OBJECTIVEs are those OBJECT-TYPEs or OBJECT-ITEMs that are specified to be (or excluded from) the focus of an ACTION.



Figure 16. Basic ACTION Structure

## SEMANTIC WEB TECHNOLOGIES

This part of the report provides a basic comprehension of Semantic Web and its latest technologies. Specifically, an examination of Web Ontology Language (OWL) and an evaluation of ontology tools are outlined in this chapter.

### Semantic Web

At a surprisingly accelerated rate, the Internet has become the central information station for individual to consume. The current web architecture contains vast amount of information resources that can be utilize in numerous beneficial manners. This web information is displayed with the aid of markup languages, such as HTML. Unfortunately, it is rendered solely for humans to translate and share data, rather than computer applications. It neither provides any meaning to this data nor help in understanding the content for computers to process. This data found on the Internet lacks structure and explicit meaning, creating difficulties for information retrieval and readability by computers. As a result, the true potential and effectiveness of the Internet is limited since it relies on human interpretation to use this information.

Therefore, the W3C organization has decided to pursue a “Semantic Web” project, where the Internet is transformed into a machine-interpretable network. It would contain “information [which has] well-defined meaning, better enabling computers and people to work in cooperation”. In this ‘semantic’ vision, the Internet would be extended with conceptual metadata[[19]](#footnote-19) that reveals the intended meaning of Web resources, making them more useful to machines. In other words, the documents are ‘marked up’ with semantic information so that the content of the document is easily interpreted by the computers. In a semantically enabled system, the tags in the document refer to defined concepts, and the system can parse the definition of the concept and use that to combine the information with other potentially related data in other applications. Subsequently, the software agents can apply this information to perform advanced tasks that humans may not be able to perform. It will “weave together an incredibly large network of human knowledge and will complement it with machine processability”. This process may ultimately create truly knowledgeable systems with various specialized reasoning services.

The Semantic Web addresses the shortcomings of the current web by offering a data centric markup language, XML, and the descriptive standards, RDF and OWL. eXtensible Markup Language (XML) provides a surface syntax for structured documents, but it does not provide sufficient data meaning for “efficient sharing of conceptualization”. In other words, it stores and displays information, however, it does not provide any description to it. Resource Description Framework (RDF) is a basic ontology language with graphical applications that combines XML syntax and semantics to represent information about resources on the web. Resources are described in terms of properties and property values using RDF statements. Due to RDF’s drawbacks listed below, this graphical language may not be a proper instrument for the Situational Awareness Ontology:

* It is weak to describe the resources in sufficient details, such as no localized range and domain constraints.
* It is difficult to provide reasoning support.

OWL (Web Ontology Language) has “more facilities, [such as additional vocabulary], for expressing meaning and semantics than XML, RDF, and RDF Schemas, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web”. In other words, OWL is a stronger language with greater machine interpretability and larger vocabulary than RDF.

To facilitate the exchange of data between computer applications, standard vocabularies of a domain must be established and captured in an ontology. It is a knowledge representation model defined in terms of classes, properties and relationships for individuals who need to share information in a domain.

Ontologies compose the primary foundation of OWL, hence it is vital to comprehend ontologies and its OWL components.

### Ontology

In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain, and may be used to describe the domain.

In theory, an ontology is a "formal, explicit specification of a shared conceptualization". An ontology renders shared vocabulary and taxonomy, which models a domain — that is, the definition of objects and/or concepts, and their properties and relations.

Ontologies are the structural frameworks for organizing information and are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, biomedical informatics, library science, enterprise bookmarking, and information architecture as a form of knowledge representation about the world or some part of it. The creation of domain ontologies is also fundamental to the definition and use of an interoperability framework.

Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. As mentioned above, most ontologies describe individuals (instances), classes (concepts), attributes, and relations. In this section each of these components is discussed in turn.

Common components of ontologies include:

* Individuals: instances or objects (the basic or "ground level" objects)
* Classes: sets, collections, concepts, classes in programming, types of objects, or kinds of things
* Attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have
* Relations: ways in which classes and individuals can be related to one another
* Function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement
* Restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as input
* Rules: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form
* Axioms: assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of "axioms" in generative grammar and formal logic. In those disciplines, axioms include only statements asserted as a priori knowledge. As used here, "axioms" also include the theory derived from axiomatic statements
* Events: the changing of attributes or relations

Ontologies are commonly encoded using ontology languages such as OWL and RDFS. OWL is W3C’s latest Semantic technology that builds these ontologies to enable agents to exchange data across web applications and resources.

### Web Ontology Language (OWL)

Released in February 2004 by the W3C, Web Ontology Language (OWL) is an ontology language that describes the classes, properties and relations between them that are inherent in Web documents and resources. Jim Hendler and Guus Schreiber, co-chairs for the Web Ontology Working Group, describe OWL as the following:

*"OWL takes a major step forward in representing and organizing knowledge on the World Wide Web. It strikes a sound balance between the needs of industry participants for a language which addresses their current Web use cases, and the restrictions on developing an ontology language that meshed with established scientific principles and research experience."*

OWL is used to describe, share and publish the set of terms that are inherent in Web documents and applications. OWL uses both Unique Resource Locators (URL)[[20]](#footnote-20) (e.g.: <http://www.w3c.org>) for naming and the description framework for the Web provided by RDF to extend the capabilities of ontologies. OWL is a vocabulary extension of RDF and RDF-S by providing an elaborated description of classes, properties, and individuals. This feature enhances the machine interpretability of Web content. OWL is derived from two other languages, DAML (DARPA Agent Markup Language) & OIL (Ontology Inference Layer).

OWL has three sub languages, each with a different level of expressive description of the data:

* OWL Lite: It is the simplest language for ontologies with simple class hierarchies and constraints. This subset of OWL-DL contains an easier reasoner than the other species.
* OWL-DL: It corresponds to Description Logics[[21]](#footnote-21), meaning that it has “decidable reasoning”. Thus, it automatically computes the classification hierarchy and checks for inconsistencies. OWL-DL does not allow datatype properties to be transitive, symmetric, or have inverse properties. Therefore, relationships can only be formed between individuals or between an individual and a data value.
* OWL Full: It is an extension of RDF with OWL syntax, where it allows for classes as instances. In OWL-Full, classes can be related, but this cannot be reasoned with.

An OWL ontology is a network of classes, properties, and individuals. Table 3 illustrates the OWL constructs used to describe the components of an ontology:

Table 3 Six core class descriptions and their definitions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Core Class** | **Class Description** | **Definition** | **Symbol** | **Meaning** | **OWL Element** |
| ***Named Class*** |  | All the individuals are part of this named class |  |  | owl:Class |
| ***Intersection Class*** | *Intersection; Conjunction* | The individuals that are contained in the overlap portion of 2+ classes | ∩ | “And” | owl:intersectionOf |
| ***Union Class*** | *Union; Disjunction* | The individuals contained in the combination of 2+ classes | ∪ | “Or” | owl:unionOf |
| ***Complement Class*** | *Negation* | The individuals that are not in the negated class | ¬ | “Not” | owl:complementOf |
| ***Restrictions*** | *Universal* | It describes the class of individuals that have only one kind of relationship along a specified property to an individual that is a member of a specified class | ∀ | “Only”; “All values from” | owl:allValuesFrom |
| *Existential* | It describes the class of individuals that have at least one kind of relationship along a specified property to an individual that is a member of a specified class | ∃ | “Some Value from”; “At least one” | owl:someValuesFrom |
| *Absolute Cardinality* | It defines the exact number of relationships that a class of individuals can have. | = | “Exactly n” | owl:cardinality |
| *Max Cardinality* | It defines the maximum number of relationships that a class of individuals can have. | ≤ | “At most n” | owl:maxCardinality |
| *Min Cardinality* | It defines the minimum number of relationships that a class of individuals can have. | ≥ | “At least n” | owl:minCardinality |
| *hasValue* | It specifies the class of individuals that participate in a specified relationship with a specific individual | ∍ | “equals x” | owl:hasValue |
| ***Enumeration Class*** |  | It specifies explicitly and exhaustively the list of the individuals that are members of the enumeration class | { ... } |  | owl:oneOf |

The OWL vocabulary provides two ‘extreme’ predefined classes, which are used to state all or no instances. An empty ontology contains one class representing the set containing all individuals has been defined as owl:Thing. New classes created will be the subclasses of this root class. The owl:Nothing class is a empty class that has no member individuals.

The relationships between the classes are defined in two manners:

* Subsumption: There is a hierarchical relationship, where ‘subclass’ and ‘superclass’ terms are applied. All subclass members can be members of the superclass.
* Disjointness: The classes that do not overlap or do not have any instances in common are known as ‘disjoint’ classes.

A property provides an association that relates an instance to a value. The following represents the categories of properties:

* Object properties (owl:ObjectProperty): Object properties link classes to classes. They may have an inverse property (e.g. the inverse of “worksFor” might be “employs”).
* Datatype properties (owl:DatatypeProperty): Datatype properties link classes to datatype values (e.g. integers, floats, strings).
* Annotation properties (owl:AnnotationProperty): Annotation properties describes instances by adding information to classes, individuals, and object/datatypes properties.
* Ontology properties (owl:OntologyProperty): Ontology properties relate ontologies to ontologies.

In OWL, properties can be specified as domain and range. For example, the property ‘hasTopping’ will allow the class ‘Pizza’ to select a range of toppings from the class ‘PizzaTopping’. Additionally, they may have sub properties leading to a hierarchy of properties. Certain property characteristics can be specified as the following:

* Functional: The functional property of an individual has a single value. For example, there can be at most one individual that is related to the individual via the property.
* Inverse Functional: The inverse of the property is functional. For example, a property links Individual A to Individual B, then its inverse property will link Individual B to Individual A.
* Symmetric: Symmetric property can relate property values back to subject resources. For example, if a property links A to B, then it can be inferred that it links B to A.
* Transitive: The transitive property allows elements in a chain to be equal. For example, if a property P links A to B and B to C, then it can be inferred that it links A to C via property P. A property that is transitive cannot be functional.

One of the appealing features of OWL is its reasoning power. Reasoning capabilities, such as consistency checking and classification, are used to detect logical inconsistencies within the ontology. OWL-DL requires a reasoner to infer information that isn’t explicitly represented in an ontology. The reasoner can check whether or not all of the statements and definitions in the ontology model are mutually consistent and can also recognize which concepts fit under which definitions. Standard ‘reasoning services’ are:

* Subsumption testing
* Equivalence testing
* Instantiation testing
* Consistency testing

Classification is used to infer specialization relationships between classes from their formal definitions. A classifier takes a class hierarchy including the logical expressions, and then returns a new class (inferred) hierarchy, which is logically equivalent to the input (asserted) hierarchy. Asserted hierarchy would represent the ‘ideal’ tree of disjoint primitives. Inferred hierarchy displays the polyhierarchy tangle (several classes with multiple parents), the ‘reality’ of the ontology. The reasoning feature adds great value in the domain of healthcare since it is consist of nested hierarchies and multi-relationships between various medical concepts. Using OWL, ontology developers could just add a new concept by describing its logical characteristics, and the classifier would automatically place it in its correct position. They can help tremendously in the construction and maintenance of large clinical terminologies.

In OWL, the reasoner applies the Open World Assumption, which means that there exists more information than presented. The reasoner cannot assume something doesn’t exist until it is explicitly stated that it does not exist. In this case, it will assume ‘the knowledge hasn’t been added to the knowledge base’.

OWL provides a promising platform for healthcare ontology and Semantic Web projects. With its strengths in reasoning capabilities and others, OWL ontologies are “more compact, less error-prone, and easier to maintain”. Hence, this robust and reusable model will play a key role in the evolution and sharing healthcare knowledge and wisdom.

## Web Services

A Web service is in fact a RPC (remote procedure call); however the vital point is that the entire world has agreed on the standard. The standard is based on the SOAP and XML-messaging. Thus, the web services technologies do not depend on any operating systems or platform. SOAP forms the layer for the messaging and XML is the formats of the messages. The WSDL file contains the technical description of the web services, such as input and output formats.

Interactions among Web services involve three types of participants: service provider, service registry and service consumer. Service registries are searchable repositories of Web Service descriptions. There are two well-known service registries: Electronic Business XML (ebXML[[22]](#footnote-22)) Registries and the Universal Description, Discovery, Integration framework (UDDI[[23]](#footnote-23)) Registries.

In addition to building web services interfaces to existing applications, there must also be a standard approach to connecting these web services together to form more meaningful business processes. The ability to integrate and assemble individual Web services into standards-based business processes is an important element of the service-oriented enterprise and the overall Web service technology “stack.”

The term Web services has gained a lot of momentum. Many software vendors (large and small) are announcing Web services initiatives and adoption. Many organizations are involved in the refinement of Web services standards. Although there seems to be a slow convergence towards a common understanding of what the term means, there is no single, universally adopted definition of what is meant by the term Web service. This situation is reminiscent of the early days of object-oriented programming: Not until the concepts of inheritance, encapsulation, and polymorphism were well defined did object-oriented programming become accepted into the mainstream of development methodologies.

Several major Web services infrastructure providers have published their definitions for a Web service:

IBM offers this definition at:

“A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging. Web services fulfill a specific task or a set of tasks. A Web service is described using a standard, formal XML notion, called its service description that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.”

The nature of the interface hides the implementation details of the service so that it can be used independently of the hardware or software platform on which it is implemented and independently of the programming language in which it is written. This allows and encourages Web services based applications to be loosely coupled, component-oriented, cross-technology implementations. Web services can be used alone or in conjunction with other Web services to carry out a complex aggregation or a business transaction.

Microsoft has a couple of definitions for Web service:

“A Web service is a unit of application logic providing data and services to other applications. Applications access Web services via ubiquitous Web protocols and data formats such as HTTP, XML, and SOAP, with no need to worry about how each Web service is implemented. Web services combine the best aspects of component-based development and the Web, and are a cornerstone of the Microsoft .NET programming model.”

A Web service is programmable application logic accessible using standard Internet protocols. Web services combine the best aspects of component-based development and the Web. Like components, Web services represent black-box functionality that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web services are not accessed via object-model-specific protocols, such as the distributed Component Object Model (DCOM), Remote Method Invocation (RMI), or Internet Inter-ORB Protocol (IIOP). Instead, Web services are accessed via ubiquitous Web protocols and data formats, such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML). Furthermore, a Web service interface is defined strictly in terms of the messages the Web service accepts and generates. Consumers of the Web service can be implemented on any platform in any programming language, as long as they can create and consume the messages defined for the Web service interface.

Sun provides the following definition at:

“Web services are software components that can be spontaneously discovered, combined, and recombined to provide a solution to the user's problem/request. The Java™ language and XML are the prominent technologies for Web services.”

It should be noted that there is broad agreement on what a Web service might be, but no single agreed-upon definition. Many developers will claim that they cannot define what a Web service is, but they know one when they see one.

In general, a Web service is a platform and implementation independent software component that can be:

* + Described using a service description language
  + Published to a registry of services
  + Discovered through a standard mechanism (at runtime or design time)
  + Invoked through a declared API, usually over a network
  + Composed with other services

One important point is that a Web service need not necessarily exist on the World Wide Web. This is an unfortunate historical naming issue. A Web service can live anywhere on the network, Inter- or intranet; some Web services can be invoked by a simple method invocation in the same operating system process, or perhaps using shared memory between tightly coupled processes running on the same machine. In fact, Web services have little to do with the browser-centric, HTML-focused World Wide Web. Sometimes, the names we choose in the information technology (IT) industry don't make a lot of sense; they simply take on a life of their own.

Another important point is that a Web service's implementation and deployment platform details are not relevant to a program that is invoking the service. A Web service is available through its declared API and invocation mechanism (network protocol, data encoding schemes, and so on). This is analogous to the relationship between a Web browser and a Web application server: Very little shared understanding exists between the two components. The Web browser doesn't particularly care if the Web application server is Apache Tomcat, Microsoft IIS, or IBM Websphere. The shared understanding is that they both speak HTTP and converse in HTML or a very limited set of MIME types. Similarly, the Web application server really doesn't care what kind of client is using it—various brands of Web browsers or even non-browser clients. This minimal shared understanding between components allows Web services to form a system of loosely coupled components.

To a business person, the Web services approach is all about integration: integrating application functionality within an organization or integrating applications between business partners (in a supply chain, for example). The important point is that application integration is enabled without tight lock-in to any particular business partner. If another supplier has a better price, shipping terms, or quality assurance, then a company's reorder systems can be easily repositioned to use that supplier; doing so is as easy as pointing a Web browser at a different Web site. With a broader adoption of Web services and XML document format standards, this style of dynamic business partner integration will become more broadly used.

When systems are this easy to integrate, an organization's reach to suppliers, customers, and other business partners is extended, yielding cost savings, flexible business models, better customer service, higher customer retention, and so on. Just as IT is fundamental to the efficient operations of an organization, Web services-based systems integration will be fundamental to flexible, lightweight systems integration—for internal application integration within an organization over an intranet and external partner integration over the Intranet or extended virtual private network. So, from a business perspective, a Web service is a business process or step within a business process that is made available over a network to internal and/or external business partners to achieve some business goal.

From a technical perspective, a Web service is nothing more than a collection of one or more related operations that are accessible over a network and are described by a service description. At this level, the Web services concept is not new. With Web services, the IT industry is trying to address the fundamental challenge of distributed computing that has been around for decades locating and accessing remote systems. The big difference is that now the industry is approaching this problem using open technology (XML and Internet protocols) and open standards managed by broad consortia such as the World Wide Web Consortium (W3C, which manages the evolution of the SOAP and WSDL specifications).

As described above, a standardized XML messaging and not being tied to any one operating system or programming language is vital for the web services. This can be visualized in the figure below.

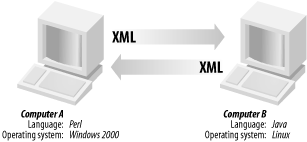


Figure 17 A Basic Web Service

There are several alternatives for XML messaging. For example, it could be used XML Remote Procedure Calls (XML-RPC) or SOAP. Alternatively, it could be just used HTTP GET/POST and pass arbitrary XML documents. Any of these options can work.

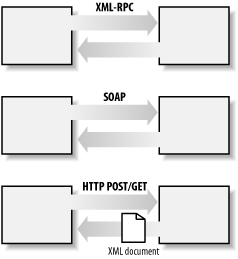


Figure 18 XML Messaging for Web Services

Although they are not required, a web service may also have two additional (and desirable) properties:

A web service should be self-describing. If you publish a new web service, you should also publish a public interface to the service. At a minimum, your service should include human-readable documentation so that other developers can more easily integrate your service. If you have created a SOAP service, you should also ideally include a public interface written in a common XML grammar. The XML grammar can be used to identify all public methods, method arguments, and return values.

A web service should be discoverable. If you create a web service, there should be a relatively simple mechanism for you to publish this fact. Likewise, there should be some simple mechanism whereby interested parties can find the service and locate its public interface. The exact mechanism could be via a completely decentralized system or a more logically centralized registry system.

* To summarize, a complete web service is, therefore, any service that:
* Is available over the Internet or private (intranet) networks
* Uses a standardized XML messaging system
* Is not tied to any one operating system or programming language
* Is self-describing via a common XML grammar
* Is discoverable via a simple find mechanism

### Web Service Architecture

There are two ways to view the web service architecture. The first is to examine the individual roles of each web service actor; the second is to examine the emerging web service protocol stack.

**Web Service Roles**

There are three major roles within the web service architecture:

Service provider: This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service requestor: This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service registry: This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearinghouse for companies and their services.

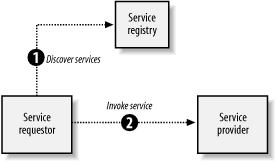


Figure 19 Web Services Role

### Web Service Protocol Stack

A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers. Following is a brief description of each layer.

**Service transport**

This layer is responsible for transporting messages between applications. Currently, this layer includes hypertext transfer protocol (HTTP), Simple Mail Transfer Protocol (SMTP), file transfer protocol (FTP), and newer protocols, such as Blocks Extensible Exchange Protocol (BEEP).

**XML messaging**

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

**Service description**

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

**Service discovery**

This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

As web services evolve, additional layers may be added, and additional technologies may be added to each layer. The figure below summarizes the current web service protocol stack.

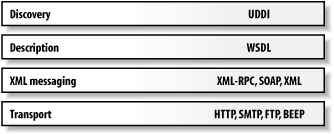


Figure 20 Web Service Protocol Stack

### Web Service Description Language (WSDL)

WSDL is a specification defining how to describe web services in a common XML grammar. WSDL describes four critical pieces of data:

* Interface information describing all publicly available functions
* Data type information for all message requests and message responses
* Binding information about the transport protocol to be used
* Address information for locating the specified service

In a nutshell, WSDL represents a contract between the service requestor and the service provider, in much the same way that a Java interface represents a contract between client code and the actual Java object. The crucial difference is that WSDL is platform- and language-independent and is used primarily (although not exclusively) to describe SOAP services.

Using WSDL, a client can locate a web service and invoke any of its publicly available functions. With WSDL-aware tools, you can also automate this process, enabling applications to easily integrate new services with little or no manual code. WSDL therefore represents a cornerstone of the web service architecture, because it provides a common language for describing services and a platform for automatically integrating those services.

WSDL is an XML grammar for describing web services. The specification itself is divided into six major elements:

**definitions**

The definitions element must be the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.

**types**

The types element describes all the data types used between the client and server. WSDL is not tied exclusively to a specific typing system, but it uses the W3C XML Schema specification as its default choice. If the service uses only XML Schema built-in simple types, such as strings and integers, the types element is not required.

**message**

The message element describes a one-way message, whether it is a single message request or a single message response. It defines the name of the message and contains zero or more message part elements, which can refer to message parameters or message return values.

**portType**

The portType element combines multiple message elements to form a complete one-way or round-trip operation. For example, a portType can combine one request and one response message into a single request/response operation, most commonly used in SOAP services. Note that a portType can (and frequently does) define multiple operations.

**binding**

The binding element describes the concrete specifics of how the service will be implemented on the wire. WSDL includes built-in extensions for defining SOAP services, and SOAP-specific information therefore goes here.

**service**

The service element defines the address for invoking the specified service. Most commonly, this includes a URL for invoking the SOAP service.

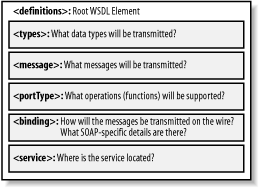


Figure 21 WSDL Elements

In addition to the six major elements, the WSDL specification also defines the following utility elements:

**documentation**

The documentation element is used to provide human-readable documentation and can be included inside any other WSDL element.

**import**

The import element is used to import other WSDL documents or XML Schemas. This enables more modular WSDL documents. For example, two WSDL documents can import the same basic elements and yet include their own service elements to make the same service available at two physical addresses.

### Simple Object Access Protocol (SOAP)

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP), for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built. This XML based protocol consists of three parts: an envelope, which defines what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing procedure calls and responses.

As an example of how SOAP procedures can be used, a SOAP message could be sent to a web-service-enabled web site such as a real-estate price database, with the parameters needed for a search. The site would then return an XML-formatted document with the resulting data, e.g., prices, location, features. With the data being returned in a standardized machine-parseable format, it can then be integrated directly into a third-party web site or application.

The SOAP architecture consists of several layers of specifications: for message format, Message Exchange Patterns (MEP), underlying transport protocol bindings, message processing models, and protocol extensibility. SOAP is the successor of XML-RPC, though it borrows its transport and interaction neutrality and the envelope/header/body from elsewhere.

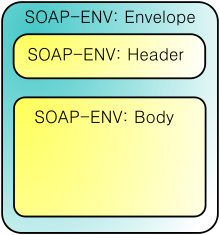


Figure 22 SOAP Message Structure

XML was chosen as the standard message format because of its widespread use by major corporations and open source development efforts. Additionally, a wide variety of freely available tools significantly eases the transition to a SOAP-based implementation. The somewhat lengthy syntax of XML can be both a benefit and a drawback. While it promotes readability for humans, facilitates error detection, and avoids interoperability problems such as byte-order, it can slow processing speed and can be cumbersome. An example message is given below:

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: 299

SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

**<?xml** version="1.0"**?>**

**<soap:Envelope** xmlns:soap="http://www.w3.org/2003/05/soap-envelope"**>**

**<soap:Header>**

**</soap:Header>**

**<soap:Body>**

**<m:GetStockPrice** xmlns:m="http://www.example.org/stock"**>**

**<m:StockName>**IBM**</m:StockName>**

**</m:GetStockPrice>**

**</soap:Body>**

**</soap:Envelope>**

**Advantages**

SOAP is versatile enough to allow for the use of different transport protocols. The standard stacks use HTTP as a transport protocol, but other protocols such as JMS and SMTP are also usable.

Since the SOAP model tunnels fine in the HTTP get/response model, it can tunnel easily over existing firewalls and proxies, without modifications to the SOAP protocol, and can use the existing infrastructure.

**Disadvantages**

Because of the verbose XML format, SOAP can be considerably slower than competing middleware technologies such as CORBA. This may not be an issue when only small messages are sent. To improve performance for the special case of XML with embedded binary objects, the Message Transmission Optimization Mechanism was introduced.

When relying on HTTP as a transport protocol and not using WS-Addressing or an Enterprise Service Bus (ESB), the roles of the interacting parties are fixed. Only one party (the client) can use the services of the other. Developers must use polling instead of notification in these common cases.

### Universal Description Discovery and Integration (UDDI)

The purpose of UDDI compliant registries is to provide a service discovery platform on the World Wide Web. Service discovery is related to being able to advertise and locate information about different technical interfaces exposed by different parties. Services are interesting when you can discover them, determine their purpose, and then have software that is equipped for using a particular type of Web service complete a connection and derive benefit from a service. A UDDI compliant registry provides an information framework for describing services exposed by any entity or business. In order to promote cross platform service description that is suitable to a “black-box” Web environment, this description is rendered in cross-platform XML.

The Universal Description, Discovery, Integration framework is jointly proposed by IBM, Microsoft and Ariba. It is a service registry architecture that presents a standard way for businesses to build a registry, discover each other, and describe how to interact over the Internet. More than 175 companies, including Boeing, Ford, CommerceOne and Sun Microsystems, have endorsed the UDDI initiative by contractually agreeing to support its future development. Conceptually, the information provided in UDDI registries consist of three components:

1. “white pages" of company contact information
2. “yellow pages" that categorize businesses by standard taxonomies
3. “green pages" that document the technical information about services

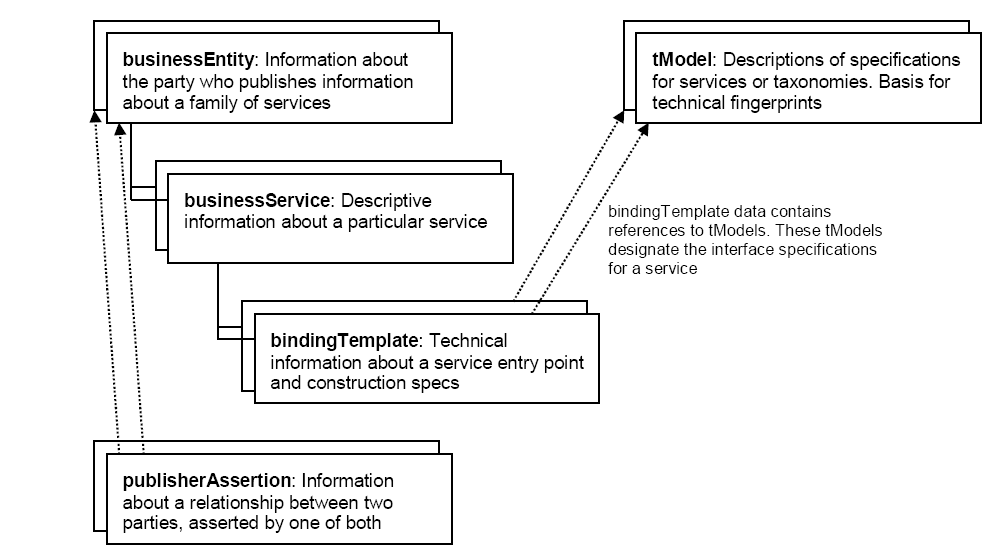


Figure 23 UDDI Core Types

The UDDI information model, defined through an XML schema, identifies five core types of information as shown in . These core types are business, service, binding, service specifications information and relationship information between two parties. Through these data structures, business entities describe information about businesses like their name, description, services offered and contact data. Business services provide more detail on each service being offered. Services can then have multiple binding templates, each describing a technical entry point for a service (e.g., mailto, http, ftp, phone, etc.). These structures use category bags for categorization purposes. An item in a category bag contains a tModel key and an associated OverviewDoc element.

The functionality provided by UDDI can be summarized as follows:

* It is possible to locate businesses and their services by their names published in the UDDI registry.
* The categories referenced in the category bags can be used to find businesses or services of a particular category. For example a user looking for a service for a particular product type can first obtain the product code from one of the defined taxonomies, like [NAICS] or [UNSPSC]. Assuming that the user wants to access the services related with optical computer disks, he obtains the UNSPSC code of “Magneto optical disks" which is “43.18.16.07.00" and searches the UDDI registry by using the APIs provided to find the businesses and their services that contain this code in their category bags. However if a business fails to provide this exact code in its category bag, it becomes impossible to locate it in this way.
* UDDI expresses the compliance of businesses and services that reference the same tModel in their descriptions.

It should be noted that a tModel is not a service description language; services need to be described in service description languages like Web Service Description Language (WSDL). There is also a need for a remote method invocation mechanism like Simple Object Access Protocol (SOAP). In the following subsection we describe the data structures in detail and how SOAP, WSDL and UDDI complement each other.

#### The businessEntity structure

The businessEntity structure represents all known information about a business or entity that publishes descriptive information about the entity as well as the services that it offers. From an XML standpoint, the businessEntity is the top-level data structure that accommodates holding descriptive information about a business or entity. Service descriptions and technical information are expressed within a businessEntity by a containment relationship.

#### The businessService structure

The businessService structures each represent a logical service classification. The name of the element includes the term “business” in an attempt to describe the purpose of this level in the service description hierarchy.

Each businessService structure is the logical child of a single businessEntity structure. The identity of the containing (parent) businessEntity is determined by examining the embedded businessKey value. If no businessKey value is present, the businessKey must be obtainable by searching for a businessKey value in any parent structure containing the businessService. Each businessService element contains descriptive information in business terms outlining the type of technical services found within each businessService element.

In some cases, businesses would like to share or reuse services, e.g. when a large enterprise publishes separate businessEntity structures. This can be established by using the businessService structure as a projection to an already published businessService. Any businessService projected in this way is not managed as a part of the referencing businessEntity, but centrally as a part of the referenced businessEntity. This means that changes of the businessService by the referenced businessEntity are automatically valid for the service projections done by referencing businessEntity structures. In order to specify both referenced and referencing businessEntity structures correctly, service projections can only be published by a save\_business message with the referencing businessKey present in the businessEntity structure and both the referenced businessKey and the referenced businessService present in the businessService structure.

#### The bindingTemplate structure

Technical descriptions of Web services are accommodated via individual contained instances of bindingTemplate structures. These structures provide support for determining a technical entry point or optionally support remotely hosted services, as well as a lightweight facility for describing unique technical characteristics of a given implementation. Support for technology and application specific parameters and settings files are also supported. Since UDDI’s main purpose is to enable description and discovery of Web Service information, it is the bindingTemplate that provides the most interesting technical data. Each bindingTemplate structure has a single logical businessService parent, which in turn has a single logical businessEntity parent.

#### The tModel structure

Being able to describe a Web service and then make the description meaningful enough to be useful during searches is an important UDDI goal. Another goal is to provide a facility to make these descriptions useful enough to learn about how to interact with a service that you don’t know much about. In order to do this, there needs to be a way to mark a description with information that designates how it behaves, what conventions it follows, or what specifications or standards the service is compliant with. Providing the ability to describe compliance with a specification, concept, or even a shared design is one of the roles that the tModel structure fills. The tModel structure takes the form of keyed metadata (data about data).

In a general sense, the purpose of a tModel within the UDDI registry is to provide a reference system based on abstraction. Thus, the kind of data that a tModel represents is pretty imprecise. In other words, a tModel registration can define just about anything, but in the current revision, two conventions have been applied for using tModels: as sources for determining compatibility and as keyed namespace references. The information that makes up a tModel is quite simple. There’s a key, a name, an optional description, and then a URL that points somewhere – presumably somewhere where the curious can go to find out more about the actual concept represented by the metadata in the tModel itself.

tModels provide the ability to describe compliance with a specification, a concept, or a shared design. When a particular specification is registered with the UDDI as a tModel, it is assigned a unique key, which is then used in the description of service instances to indicate compliance with the specification. The specification is not included in the tModel itself. The “OverviewDoc" and "OverviewURL" elements of tModels are used to point at the actual source of a specification. More precisely, the use of tModels in UDDI is two-fold:

* Defining the technical fingerprint of services: The primary role that a tModel plays is to represent a technical specification on how to invoke a registered service, providing information on the data being exchanged, the sequence of messages for an operation and the location of the service. Examples of such technical specifications include WSDL descriptions and RosettaNet PIPs [RosettaNet]. Note that the tModel mechanism describes only the signature of the services; it does not provide any information on the functionality of the service.
* Providing abstract namespace references: In UDDI, businesses, services and tModels can specify the categories to which they belong in their category bags. Categorization facilitates to locate businesses and services by relating them to some well-known industry, product or geographic categorization code set. Currently UDDI uses the North American Industrial Classification Scheme (NAICS) taxonomy for describing what a business does; the Universal Standard Products and Services Classification (UNSPSC) for describing products and services offered; and ISO 3166, a geographical taxonomy for determining where a business is located. It should be noted that any number of categories could be referenced in category bags.

It is not possible to query the attributes of tModels since they do not have any formal description. Also expressing complementary services is not possible because UDDI does not provide a mechanism to define relationships between tModels.

In UDDI, it is possible to annotate businesses and tModels with identifiers. The purpose of identifiers is to locate data more easily by using formal identifiers such as DUNS numbers, tax identifiers, or any other kind of organizational identifiers, regardless of whether these are private or shared. UDDI provides built-in tModel specifications for DUNS and Thomas register.

The UDDI specification also defines an API for interacting with UDDI registries. Inquiry APIs locate businesses, services, bindings, or tModels. Publishing APIs create or delete UDDI data in the registries.

#### The publisherAssertion structure

Many businesses, like large enterprises or marketplaces, are not effectively represented by a single businessEntity, since their description and discovery are likely to be diverse. As a consequence, several businessEntity structures can be published, representing individual subsidiaries of a large enterprise or individual participants of a marketplace. Nevertheless, they still represent a more or less coupled community and would like to make some of their relationships visible in their UDDI registrations. Therefore, two related businesses use the xx\_publisherAssertion messages, publishing assertions of business relationships.

In order to eliminate the possibility that one publisher claims a relationship between both businesses that is in fact not jointly recognized, both publishers have to agree that the relationship is valid by publishing their own publisherAssertion. Therefore, both publishers have to publish exactly the same information. When this happens, the relationship becomes visible. More detailed information is given in the appendices for the UDDI Version 2.0 API Specification.

In the case that a publisher is responsible for both businesses, the relationship automatically becomes visible after publishing just one of both assertions that make up the relationship.

The publisherAssertion structure consists of the three elements fromKey (the first businessKey), toKey (the second businessKey) and keyedReference. The keyedReference designates the asserted relationship type in terms of a keyName-keyValue pair within a tModel, uniquely referenced by a tModelKey. All three parts of the keyedReference – the tModelKey, the keyName, and the keyValue – are mandatory in this context. Empty (zero length) keyName and keyValue elements are permitted.

The following two figures ( and ) depicts an example message template for a business to register to UDDI and its corresponding instance, respectively.

**businessEntity**

**businessKey**

**name**

**URL**

**description**

**contacts**

**businessServices**

**identifierBag**

**categoryBag**

**Phone**

**Address**

**Email**

**Contact**

**businessService**

**Key**

**Name**

**Description**

**BindingTemplates**

**Phone**

**Address**

**Email**

**Contact**

**businessService**

**serviceKey**

**tModelKey**

**Name**

**Description**

**BindingTemplates**

**keyedReference**

**tModelKey**

**keyName**

**keyValue**

**keyedReference**

**tModelKey**

**keyName**

**keyValue**

**keyedReference**

**tModelKey**

**keyName**

**keyValue**

**keyedReference**

**tModelKey**

**keyName**

**keyValue**

Figure 24 Example Business Registration Template

**businessEntity**

**TB993…   
Harbour Metals**

**www.harbourmetals.co.au**

***“Serving Inner Sydney Harbour for …***

**contacts**

**businessServices**

**identifierBag**

**categoryBag**

**872-6891**

**4281 King’s Blvd, Sydney, NSW**

**Peter@harbourmetals.co.au**

**Peter Smythe**

**businessService**

**Key**

**Name**

**Description**

**BindingTemplates**

**businessService**

**23T701e54683nf…**

**Online catalog**

**“*Website where you can …***

**BindingTemplates**

**BindingTemplate**

**5E2D412E5-44EE-…**

**http://www.sydneynet/harbour…**

**tModelInstanceDetails**

**tModelInstanceInfo**

**4453D6FC-223C-3ED0…**

**http://www.rosetta.net/catalogPIP**

**keyedReference**

**DFE-2B…**

**DUNS**

**45231**

**keyedReference**

**EE123…**

**NAICS**

**02417**

tModelKeyss

Figure 25 An instance for the message template in

## DATA DISTRIBUTION SERVICE (DDS)

Many distributed applications exist today and many more are being planned for the future. One requirement common to all distributed applications is the need to pass data between different threads of execution. These threads may be on the same processor, or they may spread across different nodes. We may also have a combination of these: multiple nodes with multiple threads or processes working on each one. Each of these nodes or processes is connected through a transport mechanism such as Ethernet or shared memory. Basic protocols such as TCP/IP or higher level protocols such as HTTP can be used to provide standardized communication paths between each of the nodes. One mechanism that can be used to facilitate this data communication path is the Data Distribution Service for Real Time Systems, known as DDS.

Data Distribution Service is introduced in 2004 by Object Management Group, the company that governs specifications for CORBA, UML and many other standards. DDS is the first open international middle-ware standard directly addressing *publish-subscribe* communications for *real-time and embedded systems*. It standardizes the software application programming interface by which a distributed application can use “Data-Centric Publish-Subscribe”, the DCPS layer of DDS, as a communication mechanism. Since DDS is implemented as an infrastructure solution, it can be added as the communication interface for any software application. Today, DDS is recommended by key administration worldwide and it is widely adopted by several different application areas such as automated trading, simulations, telemetry etc. the reasons that make OMG’s Data Distribution Service as preferable as it is now can be listed as follows:

* Based on a simple Publish-Subscribe mechanism
* Flexible and adaptable architecture
* Low overhead, high-performance systems
* Deterministic data delivery
* Dynamically scalable
* Efficient use of transport bandwidth
* Usable for one-to-one, one-to-many, many-to-many communications
* Large number of quality of service (QoS) parameters that give publishers and subscribers complete control of the features of the data in the system

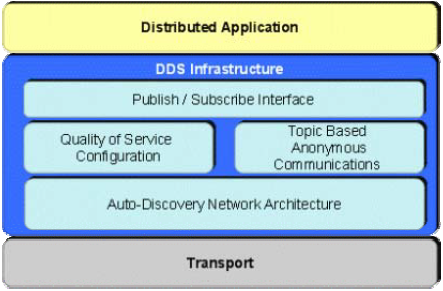


Figure 26 The structure of Data Distribution Service

As shown in , DDS provides developers with an infrastructure that enables many different applications to communicate with each other by using a very simple way.

### Publish-Subscribe Mechanism Used By DDS

Applications using publish-subscribe distribution paradigm simply communicates with each other by publishing data and subscribing the published data anonymously. In this type of communication, the only thing that a publisher needs to share its data is the name and definition of the data. A publisher does not need to know any information about a subscriber. Similarly, a subscriber retrieves data only by using the name of the data and does not have any information about the publisher of the data. As long as the interested applications know what data is being communicated, a publish-subscribe infrastructure is capable of delivering that data to the appropriate nodes.

### Data-Centric Communication

Data-centric communication means that developers have the ability to specify various features called quality of service parameters for their application. It allows developers to have a communication mechanism that is changeable to the designed application’s requirements. The rate of publication, rate of subscription, how long the data will be valid and many other parameters can be specified by the developers. The Data Distribution Service formalizes the data-centric publish-subscribe communication paradigm by providing a standardized interface.

### Data Distribution Service Components

The specification for DDS is broken up into two distinct sections. The first section covers Data-Centric Publish-Subscribe (DCPS) and the second section covers the Data Local Reconstruction Layer (DLRL).

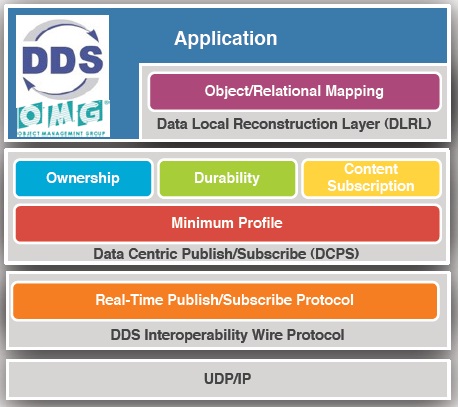


Figure 27 Layered View of Distribution Service

As it can be seen in the , DCPS is the lower layer API that an application can use to communicate with other DDS-enabled applications. DLRL is the upper layer part of the specification that outlines how an application can interface with DCPS data fields through their own object-oriented programming classes. DLRL is an optional layer within the DDS specification. Since the communication can be achieved using the DCPS layer of the DDS, it is enough to understand the components of this layer to understand the data sharing using a publish-subscribe based service. The components of the DCPS layer can be seen in the following figure:

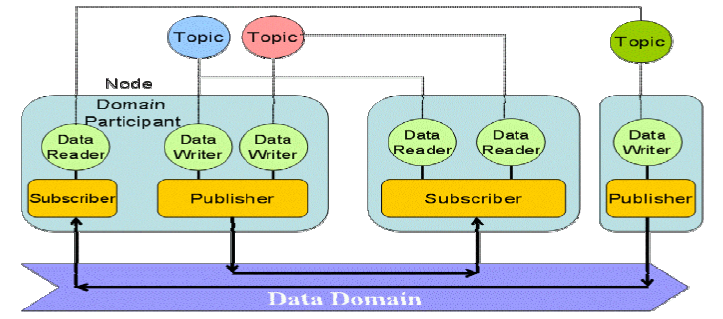


Figure 28 DCPS Entities

#### Domain and Domain Participants

The domain is the basic construct used to bind individual applications together for communication. A distributed application can elect to use a single domain for all its data-centric communications. shows a system in which six different applications are communicating within the same domain.

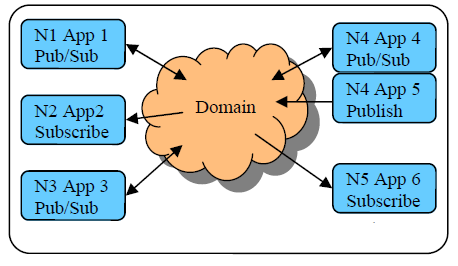


Figure 29 Single Domain System

DDS also has the capability to support multiple domains, thus it provides developers with a system that can scale with system needs or separate based on different data types. When a specific data instance is published on one domain, it will not be received by the subscribers of any other domains. In addition, multiple domains provide effective data isolation.

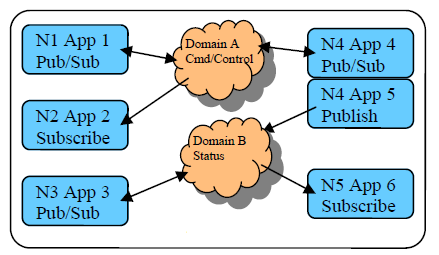


Figure 30 Multiple Domain System

An application uses an object called a “Domain Participant” to represent its activity within a domain. The Domain Participant object enables a developer to specify default QoS parameters for all data writers, data readers, publishers and subscribers in the corresponding domain. Default listener callback routines can be set up to handle events or error conditions that are reported back to the application by the DDS infrastructure. This makes it easy to specify default behavior that the application will follow when it receives a subscription for which it hasn’t set up specific listener routines for the underlying entities: Publisher, Subscriber, Data Writer or Data Reader.

#### Data Writers and Publishers

Data Writers are the primary access point for an application to publish data into a DDS data domain. Once created and configured with the correct QoS settings, an application only needs to perform a simple write call. The sending application controls the maximum rate at which data is published. Subscribers may have different requirements for how often they want to receive data. Some subscribers may want every individual sample of data, while others may want data at a much slower rate. This can be achieved by specifying a different time-based filter QoS for each subscriber. When the write function is executed, the DDS software will move the data from its current container, Data Writer, into a Publisher for sending out to the domain. shows how the entities needed to publish data are connected.

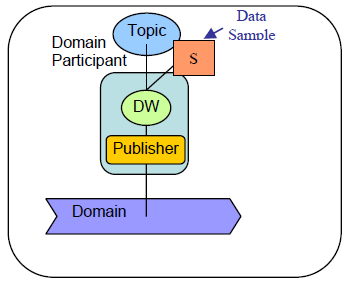


Figure 31 Publish Data

#### Data Reader and Subscribers

A Data Reader is the primary access point for an application to access data that has been received by a subscriber. Once created and configured with the correct QoS, an application can be notified that data is available in one of three ways:

• Listener Callback Routine

• Polling the Data Reader

• Conditions and WaitSets

The first method for accessing received data is to set up a listener callback routine that DDS will run immediately when data is received. You can execute your own specific software inside that callback routine to access the data. The second method is to “poll” or query the Data Reader to determine if data is available. The last method is to set up a “WaitSet”, on which the application waits until a specified condition is met and then accesses the data from the Data Reader. shows how the data retrieved from the domain.

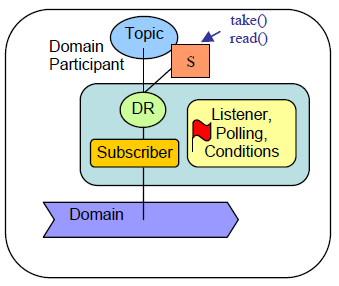


Figure 32 Subscribing Data

Having these three methods gives developers flexibility in accessing data. Accessing data is facilitated by calling take() or read() functions. The take() call removes the data from the middle-ware after returning it, while the read() allows the same data to be retrieved multiple times.

### Topic and Topic Keys

Topics provide the basic connection point between publishers and subscribers. The Topic of a given publisher on one node must match the Topic of an associated subscriber on any other node. If the Topics do not match, communication will not take place.

A Topic is comprised of a Topic Name and a Topic Type. The Topic Name is a string that uniquely identifies the Topic within a domain. The Topic Type is the definition of the data contained within the Topic. Topics must be uniquely defined within any one particular domain. Two Topics with different Topic Names but the same Topic Type definition would be considered two different Topics within the DDS infrastructure.

While defining the Topic Type, one or more data elements can be chosen as Topic Keys for that type. DDS middle-ware uses that topic type to sort the incoming data. By specifying one data element to be a key, an application can then retrieve data from the middle-ware that matches the specific key or the next key in the sequence of keys. The container holding the data of a key is considered as an Instance.

### Multi Topics and Content Filtered Topics

In addition to standard Topics, there are also constructs in place for MultiTopics and ContentFilteredTopics. A Multi Topic is a logical grouping of several Topics. It allows an application to select fields from multiple types and recombine them into a new type (something like an SQL SELECT statement). As a large application evolves, MultiTopics can be a powerful tool: the data types used within a subsystem may change, but the contents of those new types can be recombined to look like the old types, preserving the interface between subsystems. A Content Filtered Topic allows you to declare a filter expression by which only individual data samples of the specified Topic would be received and presented to the application.

### Quality of Service Parameters

Quality of service (QoS) is a significant capability provided by Data Distribution Service. The feature that enables us to specify different QoS parameters for each individual Topic, Reader or Writer gives developers a large panel from which they can design their system. This is the essence of data centricity in DDS. Some of the QoS parameters that are provided by DDS can be listed as follows:

* Deadline
* Destination Order
* Durability
* Entity Factory
* Group Data
* History
* Latency Budget
* Liveliness
* Reliability etc.

### OpenDDS[[24]](#footnote-24)

OpenDDS is an open-source C++ implementation of the Object Management Group's specification "Data Distribution Service for Real-time Systems". Although OpenDDS is itself developed in C++, Java and JMS bindings are provided so that Java applications can use OpenDDS. OpenDDS is built on the ACE abstraction layer to provide platform portability. ACE is an object oriented portability framework that implements many core design patterns for concurrent communication software. It aims to provide developers with high performance and real-time communication services and applications. It simplifies the development of object oriented network applications that utilize interprocess communications, event demultiplexing, explicit dynamic linking services into applications at run time and executing thos services in one or more processes or threads. OpenDDS also uses the capabilities of TAO[[25]](#footnote-25), which is a CORBA compliant object request broker, such as its IDL compiler and as the basis of the OpenDDS DCPS Information Repository (DCPSInfoRepo).

#### Compliance with DDS Specification

OpenDDS was originally on the 1.0 DDS Specifications, but more recent developments are based on the 1.2 DDS Specification. It currently implements all profiles, including optional ones, of the DCPS layer of DDS. None of the DLRL layer functionality is currently implemented by OpenDDS.

Developers can define a structure in IDL that will be used as a DDS data type. The structure may include basic scalar types, strings, sequences, arrays, enumeration and unions. It may not contain interfaces or value types. Zero or more keys can be specified for a data type. It also offers the following default transfer protocols:

* TCP/IP
* UDP/IP
* IP multi-cast

OpenDDS provides a pluggable transport framework that makes it easy to add a new transport.

#### Transport Options

OpenDDS supports one-to-one (point to point) and one-to-many (multi-cast) styles of publishing. The hallmark of many real-time publish-subscribe systems is the transport option which is very particular to the problem domain. Different industries already have existing transports, such as factory automation, financial trading systems, defensive systems sensors (sonar, IR, radar), etc. Many of these domains have transports combined with a higher level protocol. These legacy systems must be accommodated. OpenDDS separates the transport from the higher level protocols by means of an Extensible Transport Framework (ETF). The diagram below shows the ETF aspects of the architecture.

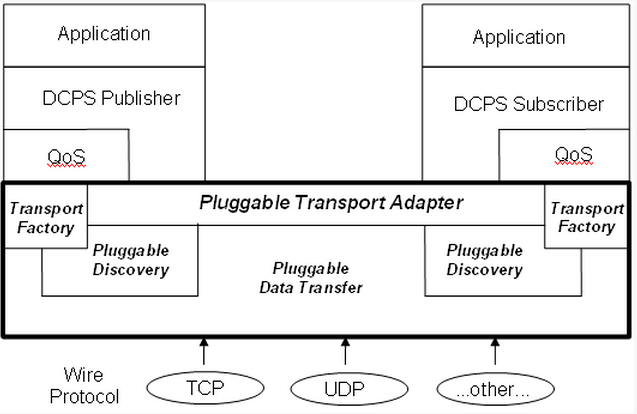


Figure 33 Pluggable Transport Framework

#### Conclusion

DDS defines a service for efficiently distributing application data between participants in a distributed application. This service is not specific to CORBA. The specification provides a Platform Independent Model (PIM) as well as a Platform Specific Model (PSM) that maps the PIM onto a CORBA IDL implementation. the service is divided into two levels of interfaces: the Data-Centric Publish-Subscribe (DCPS) layer and an optional Data Local Reconstruction Layer (DLRL). The DCPS layer transports data from publishers to subscribers according to Quality of Service constraints associated with the data topic, publisher, and subscriber. The DLRL allows distributed data to be shared by local objects located remotely from each other as if the data were local. The DLRL is built on top of the DCPS layer.

OpenDDS is the open-source C++ implementation of OMG’s DDS specification developed and commercially supported by OCI. It is available for download from <http://www.opendds.org/downloads.html>and is compatible with the latest patch levels of TAO version 1.5a, 1.6a, and the latest DOC release. OpenDDS currently implements a subset of the DCPS layer and is minimally compliant with the OMG DDS version 1.2 specifications. None of the DLRL functionality is currently implemented.

## UN/CEFACT UNQUALIFIED DATA TYPES

United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT[[26]](#footnote-26)) defined and published a catalog of data types to be used in common business document. They are generic in that they can be used as the data type of any information element. In RECONSURVE, in order to increase interoperability, these unqualified data types are adopted. The list of data types are as follows:

Table 4 Unqualified Data Types

| Primary *Representation Term* | Definition |
| --- | --- |
| Amount | A number of monetary units specified in a currency where the unit of currency is explicit or implied. |
| Binary Object | A set of finite-length sequences of binary octets.  [Note: This *Representation Term* shall also be used for *Data Types* representing graphics (i.e. diagram, graph, mathematical curves, or similar representation), pictures (i.e. visual representation of a person, object, or scene), sound, video, etc.] |
| Code | A character string (letters, figures or symbols) that for brevity and / or language independence may be used to represent or replace a definitive value or text of a *Property*.  [Note: The term 'Code' should not be used if the character string identifies an instance of an *Object Class* or an object in the real world, in which case the *Representation Term* identifier should be used.] |
| Date Time | A particular point in the progression of time (ISO 8601).  [Note: This *Representation Term* shall also be used for *Data Types* only representing a Date or a Time.] |
| Identifier | A character string used to establish the identity of, and distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme. |
| Indicator | A list of exactly two mutually exclusive Boolean values that express the only possible states of a *Property*.  [Note: Values typically indicate a condition such as on/off; true/false etc.] |
| Measure | A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN/ECE Rec. 20.  [Note: This *Representation Term* shall also be used for measured coefficients (e.g. m/s).] |
| Numeric | Numeric information that is assigned or is determined by calculation, counting or sequencing. It does not require a unit of quantity or a unit of measure.  [Note: This *Representation Term* shall also be used for *Data Types* representing Ratios (i.e. rates where the two units are not included or where they are the same), Percentages, etc.) |
| Quantity | A counted number of non-monetary units. Quantities need to be specified with a unit of quantity.  [Note: This *Representation Term* shall also be used for counted coefficients (e.g. flowers/m²).] |
| Text | A character string (i.e. a finite set of characters) generally in the form of words of a language.  [Note: This *Representation Term* shall also be used for names (i.e. word or phrase that constitutes the distinctive designation of a person, place, thing or concept).] |

## OASIS Common Alerting Protocol (CAP)

The Common Alerting Protocol (CAP)[[27]](#footnote-27) is an XML-based data format for exchanging public warnings and emergencies between alerting technologies. CAP allows a warning message to be consistently disseminated simultaneously over many warning systems to many applications. CAP increases warning effectiveness and simplifies the task of activating a warning for responsible officials. CAP also facilitates the detection of emerging patterns in local warnings of various kinds. And CAP provides a template for effective warning messages based on best practiced identified in academic research and real-world experience.

CAP provides an open, non-proprietary digital message format for all types of alerts and notifications. It does not address any particular application or telecommunications method. The CAP format is compatible with emerging techniques, such as Web services, existing formats including the Specific Area Message Encoding (SAME) user for the US National Oceanic and Atmospheric Administration (NOAA) Weather Radio and the Emergency Alert System(EAS) while offering enhanced capabilities that include:

* Flexible geographic targeting using latitude/longitude shapes and other geospatial representations in three dimensions;
  + Multilingual and multiaudience messaging;
  + Phased and delayed effective time and expiration;
  + Enhanced message update and cancellation features;
  + Template support for framing complete and effective warning messages;
  + Compatible with digital encryption and signature capability; and,
  + Facility for digital images and audio

Key benefits of CAP include reduction of costs and operational complexity by eliminating the need for multiple custom software interfaces to the many warning sources and dissemination systems involved in all-hazard warning. The CAP message format can be converted to and from all kinds of sensor and alerting technologies.

### History

The [National Science and Technology Council](http://en.wikipedia.org/wiki/National_Science_and_Technology_Council) (NSTC) report on “Effective Disaster Warnings”[PDF](http://www.sdr.gov/NDIS_rev_Oct27.pdf) (November, 2000) recommended that “a standard method should be developed to collect and relay instantaneously and automatically all types of hazard warnings and reports locally, regionally and nationally for input into a wide variety of dissemination systems.”

In 2001 an international, independent group of over 120 emergency managers began specifying and prototyping the Common Alerting Protocol data structure based on the recommendations of the NSTC report. the project was embraced by the non-profit [Partnership for Public Warning](http://www.ppw.us/) and a number of international warning system vendors. A series of field trials and long-term demonstration projects during 2002-03 led to the submission of a draft CAP specification to the OASIS standards process for formalization.

The CAP 1.0 specification was approved by [OASIS](http://en.wikipedia.org/wiki/OASIS_(organization)) in April, 2004. Based on experience with CAP 1.0, the OASIS [Emergency Management Technical Committee](http://www.oasis-open.org/committees/emergency) an Article From an Online Periodical with no DOI Assigned adopted an updated [CAP 1.1 specification](http://www.oasis-open.org/committees/download.php/14759/emergency-CAPv1.1.pdf) in October 2005. At a meeting in Geneva in October, 2006 the CAP 1.1 specification was taken under consideration by the [International Telecommunications Unio](http://en.wikipedia.org/wiki/International_Telecommunications_Union)n for adoption as an ITU recommendation.

### Applications of CAP message

The primary use of the CAP Alert Message is to provide a single input to activate all kinds of alerting and public warning systems. This reduces the workload associated with using multiple warning systems while enhancing technical reliability and target-audience effectiveness. It also helps ensure consistency in the information transmitted over multiple delivery systems, another key to warning effectiveness.

A secondary application of CAP is to normalize warnings from various sources so they can be aggregated and compared in tabular or graphic form as an aid to situational awareness and pattern detection. Although primarily designed as an interoperability standard for use among warning systems and other emergency information systems, the CAP Alert Message can be delivered directly to alert recipients over various networks, including data broadcasts. Location-aware receiving devices could use the information in a CAP Alert Message to determine, based on their current location, whether that particular message was relevant to their users.

The CAP Alert Message can also be used by sensor systems as a format for reporting significant events to collection and analysis systems and centers.

### Structure of the CAP Alert Message

Each CAP Alert Message consists of an <alert> segment, which may contain one or more <info> segments, each of which may include one or more <area> segments. Under most circumstances CAP messages with a <msgType> value of “Alert” should include at least one <info> element

* **<alert>:** It provides basic information about the current message:its purpose,its source and its status, as well as unique identifier for the current message and links to any other, related messages. It may be used alone for message acknowledgements, cancellations or other system functions, but most <alert> segments will include at least one **<info>** segment.
* **<info>:** It describes an anticipated or actual event in terms of its urgency(time available to prepare), severity(intensity of impact) and certainty(confidence in the observation or prediction), as well as providing both categorical and textual descriptions of the subject event. It may also provide instructions for appropriate response by message recipients and various other details(hazard duration,technical parameters, contact information, links to additional information sources etc.) Multiple **<info>** segments may be used to describe differing parameters or to provide the information in multiple languages.
* **<resource>:** This provides an optional reference to additional information related to the **<info>** segment within which it appears in the form of a digital asset such as an image or audio file.
* <**area>:**This describes a geographic area to which the **<info>** segment in which it appears applies. Textual and coded descriptions are supported, but the preferred ones are geospatial shapes(polygons,circles) and an altitude range,expressed in standard latitude/longitude/altitude terms in accordance with a specified geospatial datum.

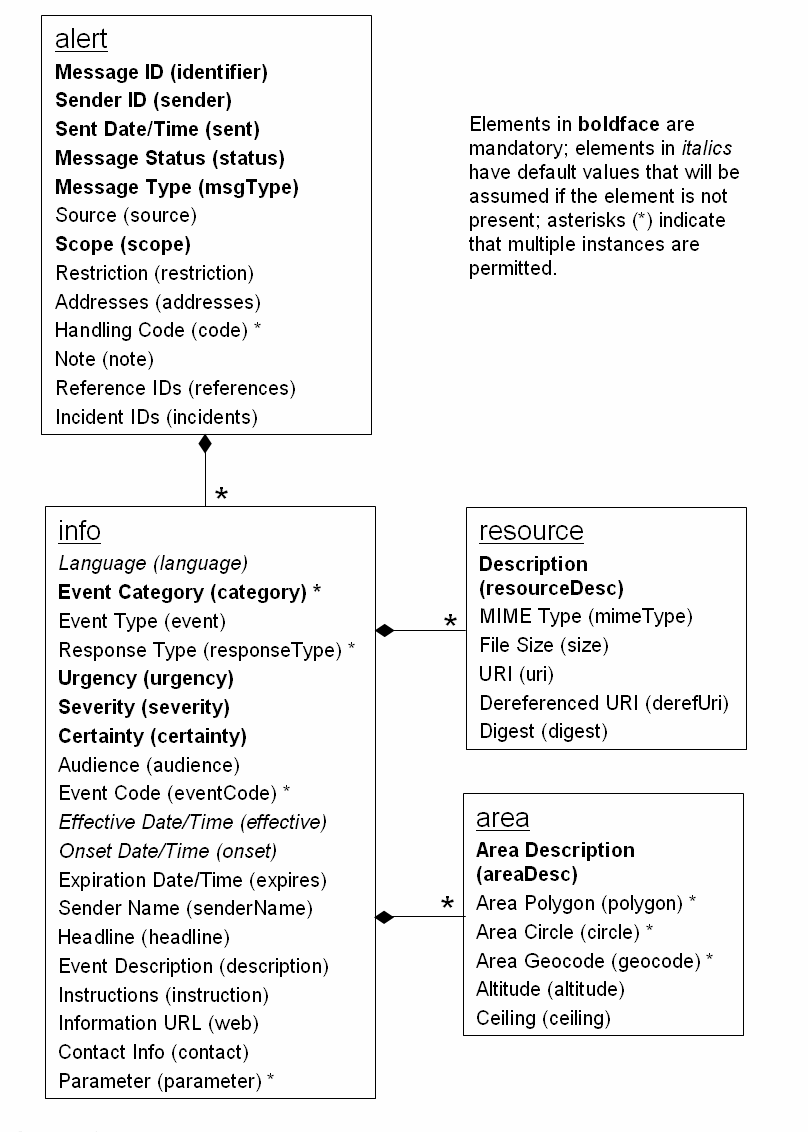


Figure 34 Document Object Model

# Conclusion

This deliverable presents the current state of the art in the standards and specifications to be used within the scope of RECONSURVE Project’s interoperability and situational awareness workpackages. The standards will be used in the following components of the RECONSURVE project as described below:

* OWL DL will be the format of the Situational Awareness Ontology.
* Racer Pro description logics reasoner will be used to maintain the Situational Awareness Ontology.
* The situational awareness rules will be defined in SWRL syntax and they will be converted to either Jess or Drools for execution.
* JC3IEDM will be the basis for the Situational Awareness Ontology and this standard will be used in the interoperability of C2 Systems.
* Web Services will be used to gather information from external organizations. This information will mainly be used to execute the situational awareness rules.
* DDS will be the mechanism to communicate with C2 systems.
* OASIS CAP will be used to send alarms to both C2 systems and external organizations.
* UN/CEFACT Unqualified Data Types will constitute the data type space of the information items exchanged in the RECONSURVE system.
* OGC SWE specifications will be used to retrieve information from sensors in a standard way.
* AIS will be used to identify the location, name, direction of the vessels.

1. <http://www.imo.org/Pages/home.aspx> [↑](#footnote-ref-1)
2. <http://www.opengeospatial.org/> [↑](#footnote-ref-2)
3. <http://www.opengeospatial.org/projects/groups/sensorweb> [↑](#footnote-ref-3)
4. <http://www.racer-systems.com/> [↑](#footnote-ref-4)
5. <http://kaon2.semanticweb.org/> [↑](#footnote-ref-5)
6. <http://owl.man.ac.uk/factplusplus/> [↑](#footnote-ref-6)
7. <http://clarkparsia.com/pellet/> [↑](#footnote-ref-7)
8. Thorsten Liebig, “Reasoning with OWL - System Support and Insights”, Technical Report,

   September 2006. [↑](#footnote-ref-8)
9. <http://www.jessrules.com/> [↑](#footnote-ref-9)
10. "Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem", Charles L. Forgy, Artificial Intelligence 19 (1982), 17-37. [↑](#footnote-ref-10)
11. <http://www.jboss.org/drools> [↑](#footnote-ref-11)
12. <http://www.jboss.org/jbpm> [↑](#footnote-ref-12)
13. <http://www.activiti.org/> [↑](#footnote-ref-13)
14. <http://www.w3.org/Submission/SWRL/> [↑](#footnote-ref-14)
15. <http://ruleml.org/> [↑](#footnote-ref-15)
16. <http://www.mip-site.org/> [↑](#footnote-ref-16)
17. <http://www.mip-site.org/publicsite/04-Baseline_3.0/JC3IEDM-Joint_C3_Information_Exchange_Data_Model/> [↑](#footnote-ref-17)
18. <http://www.mip-site.org/publicsite/03-Baseline_2.0/C2IEDM-C2_Information_Exchange_Data_Model/> [↑](#footnote-ref-18)
19. Metadata is “data about data”, meaning a set of facts about the content. [↑](#footnote-ref-19)
20. URI is an address for a resource available in the Web . [↑](#footnote-ref-20)
21. Description Logic focuses on descriptions to express logic (such as union, intersection and negation) of a domain. It emphasizes on the use of classification and subsumption reasoning for inference [11]. [↑](#footnote-ref-21)
22. www.ebxml.org [↑](#footnote-ref-22)
23. www.uddi.org [↑](#footnote-ref-23)
24. <http://www.opendds.org/Article-Intro.html> [↑](#footnote-ref-24)
25. [<http://www.theaceorb.com/product/aboutace.html>](http://www.opendds.org/Article-Intro.html) [↑](#footnote-ref-25)
26. <http://www.unece.org/cefact/about.htm> [↑](#footnote-ref-26)
27. E.Jones & A.Botterell(Eds.).(2005).Common Alerting Protocol,v.1.1. Retrieved 27th July 2011 from <http://www.oasis-open.org/committees/download.php/15135/emergency-CAPv1.1-Corrected_DOM.pdf> [↑](#footnote-ref-27)