Open Geospatial Consortium Inc.

Date: 2007-08-10

Reference number of this OGC® project document: **OGC 07-097**

Version: 2 (Rev 2.1)

Category: OGC® Best Practice

Editor: **Thomas Usländer (Ed.)**

Reference Model for the ORCHESTRA Architecture (RM-OA) V2 (Rev 2.1)

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	Abstract Specification
Document subtype:	OGC® Best Practice
Document stage:	Approved
Document language:	English

Preamble to the "Reference Model for the Orchestra Architecture (RM-OA)"

This document specifies the Reference Model for the ORCHESTRA Architecture (RM-OA). It is an extension of the OGC Reference Model and contains a specification framework for the design of geospatial service-oriented architectures and service networks. The RM-OA comprises the generic aspects of service-oriented architectures, i.e., those aspects that are independent of the risk management domain and thus applicable to other application domains. The ORCHESTRA Architecture is a platform-neutral (abstract) specification of the informational and functional aspects of service networks taking into account and evolving out of architectural standards and service specifications of ISO, OGC, W3C and OASIS. The target audience of the RM-OA comprise system architects, information modellers and system developers.

The present revision 2.1 of the RM-OA is an editorial update of revision 2.0 (OGC Discussion Paper 07-024). It is restricted to the core document of the RM-OA. The major changes comprise:

- refactoring of some basic service descriptions into interfaces according to the ORCHESTRA meta-model for services
- update of the specification of the Engineering Viewpoint
- update of the service interaction patterns, e.g. semantic catalogue and a new pattern for Geo Rights Management
- removal of information that is specific to the management of the ORCHESTRA project

The UML specification of the ORCHESTRA Meta-model for information and services is an integral part of this document.

For the ORCHESTRA abstract service specifications see http://www.eu-orchestra.org .

FP6-511678

ORCHESTRA

Open Architecture and Spatial Data Infrastructure for Risk Management

Integrated Project

Priority 2.3.2.9 Improving Risk Management

Reference Model for the ORCHESTRA Architecture (RM-OA)

Deliverable D3.2.3 RM-OA Version 2

Date: 2007-08-10

Revision: 2.1

Start date of the ORCHESTRA project:	2004-09-01
Duration of the ORCHESTRA project:	3 years
Organisation name of lead contractor for this deliverable:	Fraunhofer IITB

Document Control Page

Title	Reference Model for the ORCHESTRA Architecture (RM-OA) OGC 07-097 ORCHESTRA Deliverable D3.2.3: RM-OA Version 2 (Rev. 2.1)
Creator	Thomas Usländer, Fraunhofer IITB (Ed.) e-mail: thomas.uslaender@iitb.fraunhofer.de
Subject	ORCHESTRA Architecture Design
Description	This document specifies the Reference Model for the ORCHESTRA Architecture (RM-OA). It contains a platform-neutral specification of the ORCHESTRA Architecture and a specification framework for the design of ORCHESTRA-compliant service networks across all viewpoints.
Publisher	ORCHESTRA consortium
Contributor	Bernard, Lars — Joint Research Centre - IES Bügel, Ulrich — Fraunhofer IITB Corabœuf, Damien — BRGM Cooper, Michael — ETH Zürich Denzer, Ralf — Environmental Informatics Group Dihé, Pascal — Environmental Informatics Group Ecker, Severin — ARC Seibersdorf Research Fischer, Julian — Environmental Informatics Group Friis-Christensen, Anders — Joint Research Centre - IES Frysinger, Steve — Environmental Informatics Group Goodwin, John — Ordnance Survey Güttler, Reiner — Environmental Informatics Group Havlik, Denis — ARC Seibersdorf Research Hilbring, Désirée — Fraunhofer IITB Hofmann, Thomas — Environmental Informatics Group Holt, Ian — Ordnance Survey Humer, Heinrich — ARC Seibersdorf Research Iosifescu Enescu, Ionut — ETH Zürich Kunz, Wolfgang — Environmental Informatics Group Kutschera, Peter — ARC Seibersdorf Research Lorenzo, José — Atos Origin Spain Lutz, Michael — Joint Research Centre - IES Ma, Wenjie — Environmental Informatics Group Pichler, Guenther — Open Geospatial Consortium Europe Portele, Clemens — Open Geospatial Consortium Europe Robida, Francois — BRGM Schimak, Gerald — ARC Seibersdorf Research Schlobinski, Sascha — Environmental Informatics Group Schmieder, Martin — Fraunhofer IITB Serrano, Jean-Jacques — BRGM Sykora, Peter — ETH Zürich Usländer, Thomas — Fraunhofer IITB

Date	2007-08-10
Type	Text
Format	application/msword
Identifier	ORCHESTRA Portal: SP3 / SP3 Quality Assurance / 09: D3.2.3 / 06: D3.2.3 RM-OA V2 (2.1) – published version
Source	Not applicable
Language	en-GB.
Relation	none
Coverage	Not applicable
Rights	
Deliverable number	D3.2.3
Audience	☒ public ☐ restricted ☐ internal

Revision History

Revision	Date	Sections Changed	Description
1.10	2005-10-14	all	OGC Discussion Paper 05-107
2.0	2007-07-10	all	OGC Discussion Paper 07-024
2.1	2007-08-10	all	OGC Candidate Best Practice 07-097

Table of Contents

Figures

Tables

1 Executive Summary

Increasing numbers of natural disasters have demonstrated to the European Union the paramount importance of avoiding and mitigating natural hazards in order to protect the environment and citizens. Due to organisational and technological barriers, actors involved in the management of natural or man-made risks cannot cooperate efficiently. In an attempt to solve some of these problems, the European Commission has made “Improving risk management” one of its strategic objectives of the Information Society Technology (IST) research programme. The goal of the integrated project ORCHESTRA (Open Architecture and Spatial Data Infrastructure for Risk Management) is the design and implementation of an open, service-oriented software architecture as a contribution to overcome the interoperability problems in the domain of multi-risk management.

Public information about the ORCHESTRA project is available under http://www.eu-orchestra.org/.

The present document defines the Reference Model for the ORCHESTRA Architecture (RM-OA). The RM-OA comprises the generic aspects of service-oriented architectures, i.e., those aspects that are independent of the risk management domain and thus applicable to other application domains.

Based on a glossary of architectural terms, the RM-OA provides a specification framework for system architects, information modellers and system developers. The ORCHESTRA Architecture is a platform-neutral (abstract) specification of the informational and functional aspects of service networks taking into account and evolving out of architectural standards and service specifications of ISO, OGC, W3C and OASIS.

The structure of the RM-OA follows the viewpoints of the ISO/IEC 10746-1 Reference Model for Open Distributed Processing (RM-ODP) in the following manner:

- The RM-OA Enterprise Viewpoint provides a business perspective with respect to other European initiatives such as INSPIRE, GMES and other Integrated Projects. It yields the major architectural requirements, namely the rigorous use of standards where applicable, the independence from technology, the demand for loosely-coupled self-describing components based on a generic infrastructure and the design for change.
- The RM-OA Information Viewpoint provides a specification framework of all categories of information including their thematic, spatial, and temporal characteristics as well as their meta-information. The basic unit is the concept of a feature as an abstraction of a real world phenomenon. In principle, it follows ISO 19109 for the meta-model structure and rules of application schemas, but extends it by the pre-definition of the characteristics of eminent feature types (e.g. documents). As meta-information models are considered to be purpose-specific, the ORCHESTRA Meta-Model enables pluggable application schemas for meta-information. Furthermore, it explicitly considers the integration of data and services of existing systems (source systems) as well as the usage of ontologies.
- The RM-OA Service Viewpoint (in RM-ODP called Computational Viewpoint) specifies types of ORCHESTRA Architecture Services that support the syntactic and semantic interoperability between systems as well as the administration of service instances organised in ORCHESTRA Service Networks. The RM-OA provides textual service descriptions according to a common service description framework and contains an initial description of so-called ORCHESTRA Thematic Support services that facilitate the development of thematic functionality. Furthermore, by means of a meta-model for services on a platform-neutral level, the RM-OA provides rules how to formally specify service types based on interface types as the basic unit of re-usability and how to map them to concrete service platforms.
- The RM-OA Engineering and Technology viewpoints yield the mapping of the application schemas and service specifications to service platforms (e.g. W3C Web Services). Here, the RM-OA just provides guidance for the mapping to a given service platform and specifies engineering options for the design of ORCHESTRA Service Networks. The resulting work lead to platform-specific ORCHESTRA Implementation Specifications that are, however, documented outside of the RM-OA.

RM-OA annexes contain more detailed system requirements, a conceptual meta-information model and default application schemas for meta-information for an initial list of “purposes” (e.g. discovery).

2 Document Structure and Links

2.1 Link to the RM-OA Annexes and ORCHESTRA Deliverables

The RM-OA encompasses the results of the ORCHESTRA sub-project 3 and the related deliverables as annexes. These annexes are contained in the document package of the OGC Discussion paper 07-024.

Furthermore, they are available under http://www.eu-orchestra.org/publications.shtml#OAspecs at the Web site of the ORCHESTRA project together with the abstract specifications of the ORCHESTRA services.

Annex	Name	ORCHESTRA Deliverable	OGC Document
A	High Level Requirements Specification		
A1	Development dimensions	D3.2.1	OGC Discussion Paper 07-024
A2	System requirements	D3.2.1	OGC Discussion Paper 07-024
A3	Conceptual Meta-information model	D3.3.1	OGC Discussion Paper 07-024
B	Specification of ORCHESTRA Meta-information Models		
B1	RM-OA rules for OAS-MI	D3.3.2	OGC Discussion Paper 07-024

Table 1: Overview about the RM-OA Annexes

.

3 Introduction

3.1 Scope

This document specifies the Reference Model for the ORCHESTRA Architecture (RM-OA). It contains a specification framework for the design of ORCHESTRA-compliant service networks and provides a platform-neutral specification of the information and service viewpoints.

The RM-OA specification is structured according to the viewpoints of the Reference Model for Open Distributed Processing (RM-ODP) as defined in ISO/IEC 10746-1:1998 (E), with some modifications reflecting both ORCHESTRA needs and the design objective of a service network based on loosely-coupled components.

The RM-OA document is divided into the following sections:

- Section 4 “Glossary” provides a definition of the architectural terms used in the RM-OA.
- Section 5 “Process of the ORCHESTRA Architectural Design” describes the ORCHESTRA Reference Model resulting from the mapping of the ISO/IEC 10746-1 Reference Model for Open Distributed Processing (RM-ODP) to the ORCHESTRA architectural design process.
- Section 6 “Enterprise Viewpoint” provides a business perspective and summarises the architectural requirements for the design of ORCHESTRA-compliant service networks. The architectural requirements are motivated in detail in an argumentation chain in Annex A2 of the RM-OA..
- Section 7 “Design Decisions of the ORCHESTRA Architecture” summarises basic design decisions for the ORCHESTRA Architecture as an introduction to the architecture specification in the following section.
- Section 8 "Information Viewpoint” provides a specification framework of all categories of information dealt with by the ORCHESTRA Architecture, including their thematic, spatial, temporal characteristics as well as their meta-information.
- Section 9 “Service Viewpoint” provides a specification framework for ORCHESTRA Services. Furthermore, it contains descriptions for the services that support the syntactic and semantic interoperability between services, applications and systems as well as the administration of ORCHESTRA service networks. The description distinguishes between ORCHESTRA Architecture services that provide the generic, i.e. application-domain independent part of a service network, and ORCHESTRA Thematic Service that support particular application-domains, in the case of ORCHESTRA the risk management domain.
- Section 10 “Technology Viewpoint” describes general guidelines to be considered when specifying a platform as a service infrastructure upon which the platform-neutral ORCHESTRA Architecture may be mapped.
- Section 11 ”Engineering Viewpoint” describes topics to be considered by designers of ORCHESTRA Service Networks, in particular characteristics of ORCHESTRA Service Networks and policies w.r.t. naming of service and feature instances, discovery, user management, access control and authentication and service administration.
- Section 12 “Conclusion” lists the major aspects where the RM-OA deviates from standards. Furthermore, it provides an outlook for issues to be tackled in future RM-OA versions.

The RM-OA core document is associated with a list of annexes that provide more background information and more refined specifications. See Table 1 in section 2.1.

3.2 Intended Audience

System architects, information modellers and system developers when designing service networks taking into account relevant standards from ISO, OGC, W3C and OASIS.

3.3 References

The following references are used as background documents for the RM-OA. They are categorised in normative references (i.e. ISO Standards or respective drafts) and other technical or scientific documents and books.

3.3.1 Normative references

ISO/IEC 10746-1:1998 (E). Information technology - Open Distributed Processing - Reference model

ISO/IEC 10746-2:1996 (E). Information technology - Open Distributed Processing - Foundations

ISO/IEC TR 14252:1996. Information technology - Guide to the POSIX Open System Environment

ISO 19101:2004(E). Geographic information -- Reference model

ISO/TS 19103. Geographic information -- Conceptual schema language

ISO 19107:2004(E). Geographic information -- Spatial schema

ISO 19108:2004(E) Geographic information -- Temporal schema

ISO/FDIS 19109:2003. Text for FDIS 19109 Geographic information -- Rules for application schema, as sent to the ISO Central Secretariat for issuing as Final Draft International Standard

ISO 19111:2003(E). Geographic information -- Spatial referencing by coordinates

ISO 19112:2003(E). Geographic information -- Spatial referencing by geographic identifiers

ISO 19115:2004(E). Geographic Information -- Metadata

ISO 19119:2005. Geographic Information -- Services (see also "The OpenGIS Abstract Specification - Topic 12: OpenGIS Service Architecture" under http://www.opengis.org/docs/02-112.pdf)

ISO 19123:2005(E). Geographic Information -- Schema for coverage geometry and functions

ISO 19125-1:2004(E). Geographic Information -- Simple feature access -- Part 1: Common architecture

ISO 19136: 2007. Geographic Information -- Geography Markup Language (GML). International Standard under publication (2007-07-13)

ISO/TS 19139:2007. Geographic information -- Metadata -- XML schema implementation

3.3.2 Documents and Books

Dufourmont, H., Annoni, A., De Groof, H. (2004). INSPIRE - work programme Preparatory Phase 2005 – 2006. Publisher: ESTAT-JRC-ENV. Identifier: rhd040705WP4A_v4.5.3.doc, http://inspire.jrc.it/reports/rhd040705WP4A_v4.5.3_final-2.pdf

Egenhofer, M.J. (1989). A Formal Definition of Binary Topological Relationships. 3rd International Conference on Foundations of Data Organization and Algorithms: 457-472

GeoDRM (2006). Open Geospatial Consortium Abstract Specification 06-004r3: Geospatial Digital Rights Management Reference Model (GeoDRM RM). Version: 1.0.0. Editor: G. Vowles. 2006-02-28

GMES (2004). Global Monitoring for Environment and Security (GMES): Final Report for the GMES Initial Period (2001-2003) http://www.gmes.info/action_plan/index.html

INSPIRE (2007). Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE). http://www.ec-gis.org/inspire/directive/l_10820070425en00010014.pdf

OASIS (2006) OASIS WS-Trust 1.3 Committee Draft 01. 06 September 2006 http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-spec-cd-01.html

OMG (2006). "Software Services Profile and Metamodel". Request For Proposal OMG Document: soa/2006-09-01

ORCH-D2.1 (2006). D2.1 Final Report on User Requirements V1.4. Restricted Deliverable D2.1

Integrated Project 511678 ORCHESTRA. Editor: BRGM. 5 October 2006

ORCH-D2.3.5 (2006). Knowledge Modelling Final Report. Internal Deliverable D2.3.5 Integrated Project 511678 ORCHESTRA. Editor: Ordnance Survey. Version 1.0. 28 February 2006

ORCH-D2.4.1 (2005). Report on analysis of existing risk management processes. Deliverable D2.4.1 Integrated Project 511678 ORCHESTRA. Editor: DATAMAT. Revision [final]. 13 June 2005

ORCH-D2.4.2 (2005). Report identifying common service requirements. Deliverable 2.4.2 (2005) Integrated Project 511678 ORCHESTRA. Editor: DATAMAT. Revision [final]. 21 December 2005

ORCH-AbstrServ (2007). WP3.4 OA Service Abstract Specifications. Deliverables D3.4.x Integrated Project 511678 ORCHESTRA. Editor: Environmental Informatics Group (EIG). January 2007

ORCH-DoW (2006). Integrated Project 511678 ORCHESTRA: "Annex 1 – Description of Work". 6th Framework Programme IST Priority 2.3.2.9 Improving Risk Management. 11 May 2006

ORCH-ImplServ (2007). WP3.6 OA Service Implementation Specifications. Deliverables D3.6.x. Integrated Project 511678 ORCHESTRA. Editor: Environmental Informatics Group (EIG). 2007 (to be published)

OGC (2003). Open Geospatial Consortium Doc. No. 03-040. OGC Reference Model, Version 0.1.2 , 2003-03-04 http://portal.opengis.org/files/?artifact_id=3836

OGC (2006) Open Geospatial Consortium Discussion paper 05-087r3 "Observations and Measurements", 2006-02-24, http://portal.opengeospatial.org/files/?artifact_id=14034

Pollock, J.T., Hodgson, R. (2004). Adaptive Information. ISBN 0-471-48854-2. Wiley 2004

Powell, D. (Ed.) (1991). Delta-4: A Generic Architecture for Dependable Distributed Computing. Research Reports ESPRIT. Project 818/2252 Delta-4 Vol.1. ISBN 3-540-54985-4 Springer-Verlag 1991

RM-OA (2005) Usländer, T. (Ed.) Reference Model for the ORCHESTRA Architecture Version 1.10. Deliverable D3.2.2 of the ORCHESTRA Consortium, OGC Discussion Paper OGC 05-107 - https://portal.opengeospatial.org/files/?artifact_id=12574, October 2005

SOA-RM (2006). OASIS Reference Model for Service Oriented Architecture 1.0. Committee Specification 1, 2 August 2006. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

Studer, R.; Benjamins, V. R.; Fensel, D.: Knowledge engineering: Principles and methods. Data and Knowledge Engineering (DKE), 25(1-2):161-197, 1998.

Tomlin, C.D. (1990). Geographic Information Systems and Cartographic Modeling (Prentice-Hall)

W3C (2003). QoS for Web Services: Requirements and Possible Approaches. W3C Working Group Note, 25 November 2003, http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/

W3C (2004). Web Services Architecture. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-arch/

4 Glossary

The glossary provides the coherent terminological framework used in the RM-OA.

4.1 Abbreviations

AAA	Authentication, Authorisation, and Accounting
ACID	Atomicity, Consistency, Isolation, and Durability
CEN	Comité Européen de Normalisation (European Committee for Standardization)
CSL	Conceptual Schema Language
DIS	Draft International Standard
DoW	ORCHESTRA Description of Work
DRM	Digital Rights Management
EBAC	Expression-based access control
EC	European Commission
ESA	European Space Agency
ESDI	European Spatial Data Infrastructure
GeoRM	Rights Management related to Geographic Information
GFM	General Feature Model
GMES	Global Monitoring for Environment and Security
HCI	Human-Computer Interaction
INSPIRE	Infrastructure for Spatial Information in Europe
ID	Identifier
IS	International Standard
ISO	International Standardization Organisation
IST	Information Society Technology
LMO	Legally Mandated Organisations
OA	ORCHESTRA Architecture
OA Service	ORCHESTRA Architecture Service
OT Service	ORCHESTRA Thematic Service
OAA	ORCHESTRA Application Architecture
OAS	ORCHESTRA Application Schema
OAS-MI	ORCHESTRA Application Schema for Meta-information
OFS	ORCHESTRA Feature Set
OASIS	1) IST FP-6 project: Open Advanced System for Improved Crisis Management 2) Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
OIS	ORCHESTRA Implementation Specification
OMG	Object Management Group
OMM	ORCHESTRA Meta-model

ORCHESTRA	Open Architecture and Spatial Data Infrastructure for Risk Management
OSC	ORCHESTRA Service Component
OSI	ORCHESTRA Service Instance
OSN	ORCHESTRA Service Network
OWL	Web Ontology Language
OWL-S	Web service ontology based on OWL
RBAC	Role-based access control
RDF	Resource Description Framework
RM	Risk Management
RM-OA	Reference Model for the ORCHESTRA Architecture
RM-ODP	Reference Model for Open Distributed Processing
SOA	Service-oriented Architecture
SOA-RM	(OASIS) Reference Model for Service Oriented Architecture
SDI	Spatial Data Infrastructure
SDIC	Spatial Data Interest Communities
UAA	User Management, Authentication and Authorisation
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WIN	Wide Information Network for Risk Management
WP	Work package
WSMO	Web Service Modeling Ontology
XSD	XML Schema Definition

4.2 Terms and definitions

ABox

Set of description logics statements about individuals with reference to a TBox (so-called "extensional" knowledge).

Note: An example is: "Katrina" is-instance-of TropicalCyclone.

Access control

Combination of Authentication and Authorisation.

Accounting

Process of gathering information about the usage of resources by subjects.

Application [derived from http://www.opengeospatial.org/resources/?page=glossary]

Use of capabilities, including hardware, software and data, provided by an information system specific to the satisfaction of a set of user requirements in a given application domain.

Application Domain

Integrated set of problems, terms, information and tasks of a specific thematic domain that an application (e.g. an information system or a set of information systems) has to cope with.

Note: One example of an application domain is risk management.

Application Schema [ISO/FDIS 19109:2003]

Conceptual schema for data required by one or more applications.

Architecture (of a system) [ISO/IEC 10746-2:1996]

Set of rules to define the structure of a system and the interrelationships between its parts.

Authentication

Process of verifying the principal of a certain subject. In other words authentication indicates whether a subject is allowed to use a certain principal.

Within the authentication process a subject proves that it is allowed to act with the corresponding principal. Generally speaking, this proof can depend on a secret that can be, e.g.

- what somebody has (key, smart card, …)
- what somebody knows (password, …)
- what somebody is (biometrical data, …)
- the place somebody resides (certain computer, …)
- the skills of somebody (handmade signature)

The result of an authentication process is called a session.

Authorisation

Process of determining whether a subject is allowed to have the specified types of access to a particular resource. This is done by evaluating applicable access control information contained in a so called authorisation context.

Usually, authorisation is carried out in the context of authentication. Once a subject is authenticated, it may be authorised to perform different types of access.

Catalogue [derived from http://www.opengeospatial.org/resources/?page=glossary]

Collection of entries, each of which describes and points to a feature collection. Catalogues include indexed listings of feature collections, their contents, their coverages, and of meta-information. A catalogue registers the existence, location, and description of feature collections held by an Information Community. Catalogues provide the capability to add and delete entries. A minimum Catalogue will include the name for the feature collection and the locational handle that specifies where these data may be found. Each catalogue is unique to its Information Community.

Component

Hardware component (device) or Software Component.

Conceptual model [ISO/FDIS 19109:2003(E); ISO 19101]

Model that defines concepts of a universe of discourse.

Conceptual schema [ISO/FDIS 19109:2003(E); ISO 19101]

Formal description of a conceptual model.

Coverage [ISO 19123]

Function from a spatial, temporal or spatiotemporal domain to an attribute range. A coverage associates a position within its domain to a record of values of defined data types. Thus, a coverage is a feature with multiple values for each attribute type, where each direct position within the geometric representation of the feature has a single value for each attribute type.

Description Logics

Family of logic based knowledge representation languages that are a decidable subset of first order logic with well defined semantics and inferencing (problem decision procedures). In Description Logics, a distinction is made between the terminological knowledge (the so-called TBox) and the assertional knowledge (ABox). This distinction is useful for knowledge base modelling and engineering: for modelling it is just natural to distinguish between concepts and individuals; for engineering it helps by separating key inference problems, e.g. classification is related to the TBox, while instance checking is related to the ABox.

Discovery [derived from W3C: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#discovery]

Act of locating a machine-processable description of a resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions.

Engineering viewpoint

Viewpoint of the ORCHESTRA Reference Model that specifies the mapping of the ORCHESTRA service specifications and information models to the chosen service platform and the characteristics of ORCHESTRA Service Networks.

End user

Members of agencies (e.g. civil or environmental protection agencies) or private companies that are involved in an application domain (e.g. risk management) and that use the applications built by the system users according to the ORCHESTRA Architecture.

External Source System

Source system that does not provide its data and functions through an ORCHESTRA-conformant interface.

Feature [derived from ISO 19101]

Abstraction of a real world phenomenon [ISO 19101] perceived in the context of an ORCHESTRA Application.

Note: The ORCHESTRA understanding of a “real world” explicitly comprises hypothetical worlds. Features may but need not contain geospatial properties. In this general sense, a feature corresponds to an “object” in analysis and design models.

Framework [http://www.opengeospatial.org/resources/?page=glossary]

An information architecture that comprises, in terms of software design, a reusable software template, or skeleton, from which key enabling and supporting services can be selected, configured and integrated with application code.

Gazetteer [http://www.opengeospatial.org/resources/?page=glossary]

A catalogue of toponyms (place names) assigned with geographic references. A gazetteer service retrieves the geometries for one or more features, given their associated well-known feature identifiers (text strings).

Generic

A service is generic, if it is independent of the application domain. A service infrastructure is generic, if it is independent of the application domain and if it can adapt to different organisational structures at different sites, without programming (ideally).

Geospatial [http://www.opengeospatial.org/resources/?page=glossary**]**

Referring to a location relative to the Earth's surface. “Geospatial” is more precise in many geographic information system contexts than "geographic," because geospatial information is often used in ways that do not involve a graphic representation, or map, of the information.

Implementation [http://www.opengeospatial.org/resources/?page=glossary**]**

Software package that conforms to a standard or specification. A specific instance of a more generally defined system.

Information Community [http://www.opengeospatial.org/resources/?page=glossary**]**

A collection of people (a government agency or group of agencies, a profession, a group of researchers in the same discipline, corporate partners cooperating on a project, etc.) who, at least part of the time, share a common digital geographic information language and common spatial feature definitions.

Information viewpoint

Viewpoint of the ORCHESTRA Reference Model that specifies the modelling approach of all categories of information the ORCHESTRA Architecture deals with including their thematic, spatial, temporal characteristics as well as their meta-information.

Interface [ISO 19119:2005; http://www.opengis.org/docs/02-112.pdf]

Named set of operations that characterize the behaviour of an entity.

The aggregation of operations in an interface, and the definition of interface, shall be for the purpose of software reusability. The specification of an interface shall include a static portion that includes definition of the operations. The specification of an interface shall include a dynamic portion that includes any restrictions on the order of invoking the operations.

Interoperability [ISO 19119:2005 or OGC; http://www.opengeospatial.org/resources/?page=glossary]

Capability to communicate, execute programs, or transfer data among various functional units in a manner that require the user to have little or no knowledge of the unique characteristics of those units [ISO 2382-1].

Knowledge Base

Store of formal knowledge about identifiable entities of a real or hypothetical world. The entity descriptions are typically instance knowledge or data, or an ABox in terms of Description Logics. In some cases, the knowledge base additionally provides access to the knowledge schema (the TBox corresponding to the ABox). Generally, a knowledge base does not necessarily need to be described by means of a schema: it basically provides a flexible means for identification, representation and interlinking of entities. Compared to a conventional relational database, a knowledge base is more flexible: it may comprise several identifiable sets of entity relationships ("models"), and new models can dynamically be added without the need for redefining the complete database schema. New entities and relations can be inserted at run time (population of the knowledge base).

Note: Knowledge stored in a knowledge base can be retrieved by means of a query language. Compared to a Catalogue and/or a Feature Access Service (see section 9.7.1), the result of these queries is not necessarily a feature collection, e.g. just a boolean value an extreme case. If the knowledge base contains implicitly represented information, e.g. in terms of rules, the quality of the query results may be improved by automatically inferring new knowledge (TBox and/or ABox reasoning).

Loose coupling [W3C; http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#loosecoupling]

Coupling is the dependency between interacting systems. This dependency can be decomposed into real dependency and artificial dependency: Real dependency is the set of features or services that a system consumes from other systems. The real dependency always exists and cannot be reduced. Artificial dependency is the set of factors that a system has to comply with in order to consume the features or services provided by other systems. Typical artificial dependency factors are language dependency, platform dependency, API dependency, etc. Artificial dependency always exists, but it or its cost can be reduced. Loose coupling describes the configuration in which artificial dependency has been reduced to the minimum.

Meta-information

Descriptive information about resources in the universe of discourse. Its structure is given by a meta-information model depending on a particular purpose.

Note: A resource by itself does not necessarily need meta-information. The need for meta-information arises from additional tasks or a particular purpose (like catalogue organisation), where many different resources (services and data objects) must be handled by common methods and therefore have to have/get common attributes and descriptions (like a location or the classification of a book in a library).

Meta-information model

Implementation of a conceptual model for meta-information. It is represented by an ORCHESTRA Application Schema for Meta-information.

Middleware [http://www.opengeospatial.org/resources/?page=glossary]

Software in a distributed computing environment that mediates between clients and servers.

OA Info-structure Service

OA Service that is required to operate an OSN in the sense that it plays an indispensable role in the operation of an OSN.

OA Support Service

OA Service that facilitates the operation of an OSN , e.g. providing an added-value by combining the usage of OA Info-Structure Services.

Ontology [based on (Studer et al 1998)]

Explicit, formal specification of a shared conceptualisation (Studer et al 1998).

It is formal in order to not only make it readable by humans, but also by machines. It is explicit as it is based on a taxonomy specified in terms of concepts, properties (or relationships) and axioms (the "vocabulary"). It is shared in the sense that these specifications are fixed as an agreement set up and shared by a dedicated user community and that it is associated with a particular subject area (domain) or task. It is a conceptualisation as it defines a conceptual schema by abstracting from a real or hypothetical world. Its ultimate purpose is to enable machine understanding which in turn provides the potential for data and service interoperability.

In Description Logics, an ontology describes a TBox; optionally, it may also describe an ABox. The TBox can then be considered to be the schema of the ABox.

Open Architecture [based on (Powell 1991)]

Architecture whose specifications are published and made freely available to interested vendors and users with a view of widespread adoption of the architecture. An open architecture makes use of existing standards where appropriate and possible and otherwise contributes to the evolution of relevant new standards.

Operation [ISO 19119:2005; http://www.opengis.org/docs/02-112.pdf]

Specification of a transformation or query that an object may be called to execute. An operation has a name and a list of parameters.

ORCHESTRA Architecture (OA)

Open architecture that comprises the combined generic and platform-neutral specification of the information and service viewpoint as part of the ORCHESTRA Reference Model.

ORCHESTRA Application

Set of software components that together comprise an application based on the usage of ORCHESTRA Services

ORCHESTRA Application Architecture (OAA)

Instantiation of the ORCHESTRA Architecture by inclusion of those thematic aspects that fulfil the purpose and objectives of a given application. The concepts for such an application stem from a particular application domain (e.g. a risk management application).

ORCHESTRA Architecture Service (OA Service)

ORCHESTRA Service that provides a generic, platform-neutral and application-domain independent functionality.

ORCHESTRA Application Schema (OAS) [extending ISO/FDIS 19109:2003]

Conceptual schema for the data required by one or more ORCHESTRA Applications. As such it provides a formal specification that is compliant to the ORCHESTRA Meta-model of the concepts (e.g. feature types), their properties and associations which are relevant for a specific information model in an ORCHESTRA Service Network.

ORCHESTRA Application Schema for Meta-information (OAS-MI)

Form of an ORCHESTRA Application Schema applied to meta-information.

ORCHESTRA Application Implementation Specification (OAIS)

Extension and restriction of an ORCHESTRA Implementation Specification according to the needs of a particular application domain. An OAIS comprises a platform-specific combined specification of a thematic information model and a set of OT Services.

ORCHESTRA Feature Set (OFS)

Set of feature instances following the information model formally specified in an ORCHESTRA Application Schema.

ORCHESTRA Implementation Specification

Combined platform-specific specification of the engineering and technology viewpoints as a result of the mapping of the ORCHESTRA Architecture to a specific platform.

ORCHESTRA Meta-Model (OMM)

Framework of rules for the specification of ORCHESTRA Application Schemas or ORCHESTRA Service Types. It is specified in terms of UML classes stereotyped as <<MetaClass> and associated rules.

ORCHESTRA Reference Model

The ORCHESTRA Reference Model comprises a specification of all RM-ODP viewpoints for the open architecture for risk management. In particular, it encompasses the specification of the ORCHESTRA Architecture and a specification framework for ORCHESTRA Implementation Specifications which are implemented in ORCHESTRA Service Components and deployed in an ORCHESTRA Service Network as ORCHESTRA Service Instances.

ORCHESTRA Service

Service specified as an ORCHESTRA Service Type, implemented as ORCHESTRA Service Component and offered in an ORCHESTRA Service Network by an ORCHESTRA Service Instance.

ORCHESTRA Service Component

Component that provides an external interface of an ORCHESTRA Service according to an ORCHESTRA Implementation Specification.

ORCHESTRA Service Instance

Executing manifestation of an ORCHESTRA Service Component.

ORCHESTRA Service Network

Set of networked hardware components and ORCHESTRA Service Instances that interact in order to serve the objectives of ORCHESTRA Applications. The basic unit within an OSN for the provision of functions are the OSIs.

ORCHESTRA Service Type

Type of an ORCHESTRA Service specified according to the rules of the ORCHESTRA Reference Model.

ORCHESTRA Service Types are functionally classified in ORCHESTRA Architecture Services (OA Services) and ORCHESTRA Thematic Services (OT Services).

ORCHESTRA Source System

Source system that provides its data and functions through an ORCHESTRA-conformant interface. Each ORCHESTRA Source System is associated to at least one External Source System.

ORCHESTRA Thematic Service (OT Service)

ORCHESTRA Service that provides an application domain-specific functionality built on top and by usage of OA Services and/or other OT Services.

Note: An OT Service may but need not be specified in a platform-neutral way.

Purpose (of meta-information)

A purpose of meta-information describes the goal of the usage of the resources.

(Service) Platform

Set of infrastructural means and rules that describe how to specify service interfaces and related information and how to invoke services in a distributed system.

Examples for platforms are Web Services according to the W3C specifications including a GML profile for the representation of geographic information or a CORBA-based infrastructure with a UML profile according to the OMG specifications.

Principal

A principal represents the identity of a subject in an ORCHESTRA Service Network. A subject may have several identities, and thus several principals. The association between a principal and a subject is established in an authentication process.

Reference Model [ISO Archiving Standards; http://ssdoo.gsfc.nasa.gov/nost/isoas/us04/defn.html]

A reference model is a framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is based on a small number of unifying concepts and may be used as a basis for education and explaining standards to a non-specialist.

Resource

Functions (possibly provided through services) or data objects.

Semantic Web [W3C; http://www.w3.org/2001/sw/Overview.html]

The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming.

Service [ISO 19119:2005; ISO/IEC TR 14252; http://www.opengis.org/docs/02-112.pdf]

Distinct part of the functionality that is provided by an entity through interfaces.

Note: In ORCHESTRA, such an entity is called ORCHESTRA Service Component when referring to the software component and ORCHESTRA Service Instance when referring to the running instance in an ORCHESTRA Service Network.

Service Mapping

Process of mapping a description of an ORCHESTRA Service Type and the specification of its interfaces on platform-neutral level to an ORCHESTRA Implementation Specification for a given platform.

Service Profile Specification

ORCHESTRA Implementation Specification defining a functional subset of an interface of an ORCHESTRA Service Type as a result of a service mapping. The functional subset is defined in the sense that those operations and parameters that are marked on the abstract level as “mapping not required” may be omitted for the platform-specific specification.

Service Viewpoint

Viewpoint of the ORCHESTRA Reference Model that specifies the ORCHESTRA services supporting the syntactic and semantic interoperability between source systems and the development of ORCHESTRA Applications.

Session

Temporary association between a subject and a principal as a result of an authentication process initiated by the subject. Information about a session is stored in authentication session information.

Software Component [derived from component definition of http://www.opengeospatial.org/resources/?page=glossary]

Software program unit that performs one or more functions and that communicates and interoperates with other components through common interfaces.

Source System

Container of unstructured, semi-structured or structured data and/or a provider of functions in terms of services. The source systems are of very heterogeneous nature and contain information in a variety of types and formats.

Spatial Data Infrastructure [http://www.gsdi.org/pubs/cookbook/chapter01.html#spatial]

Relevant base collection of technologies, policies and institutional arrangements that facilitate the availability of and access to spatial data. The Spatial Data Infrastructure provides a basis for spatial data discovery, evaluation, and application for users and providers within all levels of government, the com-

mercial sector, the non-profit sector, academia and by citizens in general.

Subject

Abstract representation of a user or a software component in an ORCHESTRA Application.

System [ISO/IEC 10746-2:1996]

Something of interest as a whole or as comprised of parts. Therefore a system may be referred to as an entity. A component of a system may itself be a system, in which case it may be called a subsystem.

Note: For modelling purposes, the concept of system is understood in its general, system-theoretic sense. The term "system" can refer to an information processing system but can also be applied more generally.

System User

Provider of services that are used for an application domain as well as IT architects, system developers, integrators and administrators that conceive, develop, deploy and run applications for an application domain.

TBox

Describes relations between concepts (so-called "intensional" knowledge) without regarding concrete individuals.

Note: An example is: Every TropicalCyclone has-exactly 1 hurricaneCategory.

Technology viewpoint

Viewpoint of the ORCHESTRA Reference Model that specifies the technological choices of the service platform and its operational issues.

Thesaurus (Pollock, Hodgson 2004).

Synonym and antonym repository for data vocabulary terminology.

Transaction [W3C, http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#transaction]

Transaction is a feature of the architecture that supports the coordination of results or operations on state in a multi-step interaction. The fundamental characteristic of a transaction is the ability to join multiple actions into the same unit of work, such that the actions either succeed or fail as a unit.

User

Human acting in the role of a system user or end user of the ORCHESTRA Architecture.

Viewpoint [RM-ODP]

Subdivision of the specification of a complete system, established to bring together those particular pieces of information relevant to some particular area of concern during the design of the system.

Universe of discourse [ISO 19101]

View of the real or hypothetical world that includes everything of interest.

Web Service

Self-contained, self-describing, modular service that can be published, located, and invoked across the Web. A Web service performs functions, which can be anything from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.

W3C Web Service [W3C, http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice]

Software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

4.3 General Remark

This document follows the ISO/IEC Directives, Part 2: Rules for the structure and drafting of International Standards w.r.t. the usage of the word “shall”**.** The word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

5 Process of the ORCHESTRA Architectural Design

5.1 Overview

The ORCHESTRA Architecture is being designed in an iterative way recognising the fact that the requirements of the system and of the end users as well as the technological progress in the IT market and in IT standardisation have a dynamic nature and cannot be completely caught in a one-shot design. Thus, a global iteration cycle between the analysis, the design, the implementation and the deployment phase of the architecture is foreseen.

Figure 1 illustrates the iteration cycle between the analysis and the design phase which is explained further in the following paragraphs.

A **consolidation process** in-between ensures that, at a defined point in time, there is a common understanding of the system requirements, the user requirements and an assessment of the current technology as a foundation to design the ORCHESTRA Architecture.

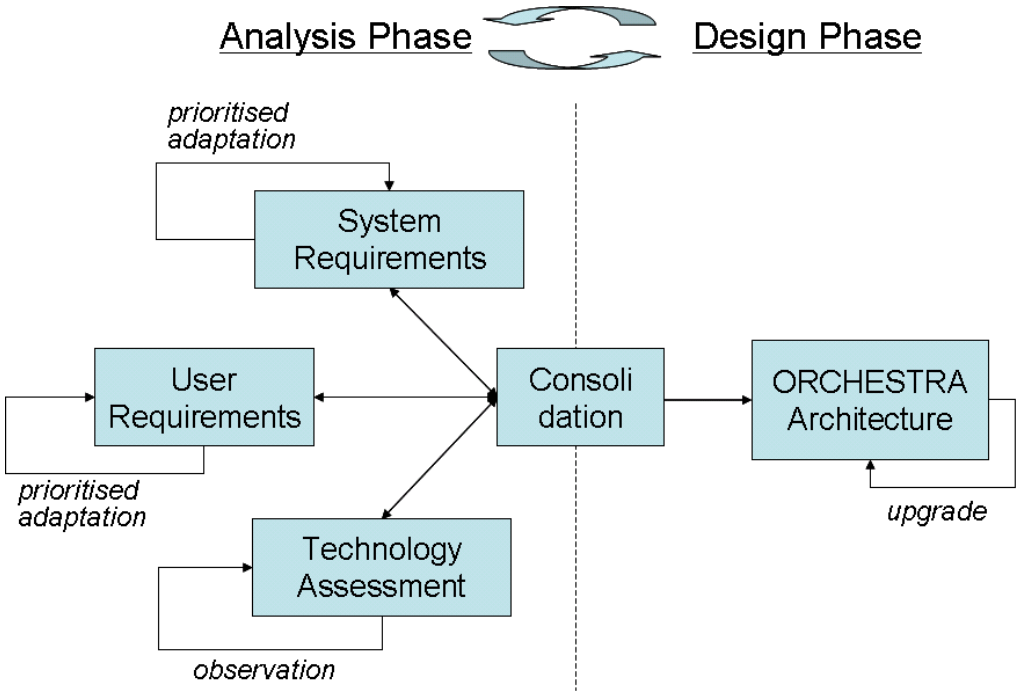

Figure 1: Dynamic ORCHESTRA Analysis and Design Process

System requirements for the ORCHESTRA Architecture encompass all functional and non-functional aspects that need to be considered in order to enable interoperability between systems. Interoperability is understood here according to ISO 19119:2005 as the capability to communicate, execute programs, or transfer data among various functional units in a manner that require the user to have little or no knowledge of the unique characteristics of those units.

Thus, system requirements for the ORCHESTRA Architecture are requirements for the infrastructure. Within the RM-OA, they originate from the combined expertise of the consortium in the area of interoperability as well as from (ORCH-DoW 2006).

Starting from a view oriented at system user roles, the system requirements for the ORCHESTRA Architecture are finally expressed in terms of architectural principals (see section A2.1.4 in the RM-OA Annex A2) that a system should follow. These architectural principals aim at improving the exchange, sharing and using of information and services among various functional units cross system boundaries, i.e. boundaries of existing systems which for some purpose need to collaborate with each other.

System requirements are expressed in generic technical terms, i.e. independent of application domains.

User requirements for the ORCHESTRA Architecture encompass all aspects that users or end-users of the ORCHESTRA Architecture expect to be reflected by a service infrastructure. User requirements are usually expressed in terms that are tailored to the needs of a specific application domain, for ORCHESTRA being the domain of risk management. As such, user requirements for the ORCHESTRA Architecture have to be aligned with and mapped to generic system requirements.

Technology assessment is a continuous process, too. ORCHESTRA aims at building the architecture on top of and abstracting from technologies, tools and products that are either standard approaches or have proven to be successful in solving interoperability problems in deployed use-cases.

The dynamic nature of these three input factors of the ORCHESTRA Architecture naturally leads to an iterative architectural design process. Various but controlled upgrades of the ORCHESTRA Architecture are required to adapt the architecture to the changing needs. Both the system and the user requirements are dynamic in the sense that they will be prioritised and adapted in local iteration cycles. A **consolidation** process is required in order to assess them in the light of time, budget and technological constraints. The consolidation process is determined by the answers to the following questions:

- How can the user requirements be realised by generic concepts such that a re-use for other application domains will be possible ?
- Which user requirements are of utmost importance with respect to the pilot scenarios in which the ORCHESTRA results are to be validated in a first place?
- What is the status of the existing technology in order to realise a given user requirement ?
- What is the effort to realise a user requirement in a given environment ?

As constant factors across the ORCHESTRA architectural design process, ORCHESTRA follows in each iteration step the principles of the Reference Model for Open Distributed Processing (RM-ODP) and the taxonomy of the ORCHESTRA services as described in subsections 5.2 and 5.4.

5.2 Application of the Reference Model of Open Distributed Processing (RM-ODP)

5.2.1 RM-ODP Overview

The Reference Model of Open Distributed Processing (ISO/IEC 10746-1:1998) is an international standard for architecting open, distributed processing systems. It provides an overall conceptual framework for building distributed systems in an incremental manner. The RM-ODP standards have been widely adopted: they constitute the conceptual basis for the ISO 19100 series of geomatics standards (normative references in ISO 19119:2005), and they also have been employed in the OMG object management architecture.

The RM-ODP approach has been used in the design of the OpenGIS Reference Model (OGC 2003) with respect to the following two aspects:

- It constitutes a way of thinking about architectural issues in terms of fundamental patterns or organizing principles, and
- It provides a set of guiding concepts and terminology.

Systems resulting from the RM-ODP approach (called ODP systems) are composed of interacting objects (see section 7.1.1 of ISO/IEC 10746-1:1998) whereby in RM-ODP an object is a representation of an entity in the real world. It contains information and offers services.

Based on this understanding of a system, ISO/IEC 10746 specifies an architectural framework for structuring the specification of ODP systems in terms of the concepts of viewpoints and viewpoint specifications, and distribution transparencies.

The viewpoints identify the top priorities for architectural specifications and provide a minimal set of requirements—plus an object model—to ensure system integrity. They address different aspects of the system and enable the ‘separation of concerns’.

Five standard viewpoints are defined:

- The enterprise viewpoint: A viewpoint on the system and its environment that focuses on the purpose, scope and policies for the system.
- The information viewpoint: A viewpoint on the system and its environment that focuses on the semantics of the information and information processing performed.
- The computational viewpoint: A viewpoint on the system and its environment that enables distribution through functional decomposition of the system into objects which interact at interfaces.
- The engineering viewpoint: A viewpoint on the system and its environment that focuses on the mechanisms and functions required to support distributed interaction between objects in the system.
- The technology viewpoint: A viewpoint on the system and its environment that focuses on the choice of technology in that system.

The aspect of a distributed ODP system is handled by the concept of distribution transparency. Distribution transparency relates to the masking from applications the details and the differences in mechanisms used to overcome problems caused by distribution. According to the RM-ODP, application designers simply select which distribution transparencies they wish to assume and where in the design they are to apply. The RM-ODP distinguishes between eight distribution transparency types. These distribution transparencies consider aspects of object access, failure of objects, location of objects, as well as replication, migration, relocation, persistence and transactional behaviour of objects.

5.2.2 Mapping of RM-ODP to the ORCHESTRA Architectural Design Process

An RM-ODP-based approach has been selected for the design of the ORCHESTRA Architecture as the primary objectives of RM-ODP, such as

- support for aspects of distributed processing,
- provision of interoperability across heterogeneous systems, and
- hiding consequences of distribution to systems developers,

are largely coherent with the ORCHESTRA objectives. However, as an ORCHESTRA system will have the characteristic of a loosely-coupled network of systems and services instead of a “distributed processing system based on interacting objects”, the RM-ODP concepts are not followed literally. For instance, the ORCHESTRA concepts are not specified in terms of the RM-ODP distribution transparencies as these are specified in terms of interacting objects.

The usage of RM-ODP for the ORCHESTRA Architectural design process focuses on the structuring of ideas and the documentation of the ORCHESTRA Architecture. Thus, a mapping of the RM-ODP viewpoints to the ORCHESTRA needs has been applied and summarised in Table 2:

- The second column of Table 2 provides the original definitions of the viewpoints as given in the OpenGIS Reference Model using the terms of the OpenGIS glossary.
- The third column of Table 2 indicates the mapping of the viewpoints to the ORCHESTRA needs using the terms as defined in the ORCHESTRA glossary (see section 4).

  Note: In order to highlight the fact, that an ORCHESTRA deployment will have the nature of a loosely-coupled distributed system based on networked services rather than a distributed application based on computational objects, the “computational viewpoint” is referred to as “service viewpoint” in ORCHESTRA.
- The fourth column of Table 2 provides examples of what will be defined in the respective viewpoint.

Viewpoint Name	Definition according to ISO/IEC 10746	Definition according to the OpenGIS Reference Model	Mapping to the ORCHESTRA architectural design process	Examples
Enterprise	Concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part.	Focuses on the purpose, scope and policies for that system.	Reflects the analysis phase in terms of the system and the user requirements as well as the technology assessment. Includes rules that govern actors and groups of actors, and their roles.	Use case definition for a statistical processing service. Rules for the maintenance and evolution of the architecture.
Information	Concerned with the kinds of information handled by the system and constraints on the use and interpretation of that information.	Focuses on the semantics of information and information processing.	Specifies the modelling approach of all categories of information the ORCHESTRA Architecture deals with including their thematic, spatial, temporal characteristics as well as their metadata.	Information objects specified in UML class diagrams and referred to by the specification of the processing service (e.g. as parameter types).
Computational	Concerned with the functional decomposition of the system into a set of objects that interact at interfaces – enabling system distribution.	Captures component and interface details without regard to distribution.	Referred to as “Service Viewpoint” Specifies the ORCHESTRA Interface and Service Types that aim at improving the syntactic and semantic interoperability between services, source systems and ORCHESTRA Applications.	Specification of the externally visible behaviour of a service type, e.g. UML specification of the interface types of the processing service including the possibility to perform statistics Service support for service orchestration and choreography.
Technology	Concerned with the choice of technology to support system distribution.	Focuses on the choice of technology.	Specifies the technological choices of the platform, its characteristics and its operational issues.	Specification of the platform “ORCHESTRA Web Services” consisting of W3C Web Services according to (W3C 2004) and a GML profile.
Engineering	Concerned with the infrastructure required to support system distribution.	Focuses on the mechanisms and functions required to support distributed interaction between objects in the system.	Specifies the mapping of the ORCHESTRA service specifications and information models to the chosen platform. Considers the characteristics and principles for service networks.	Provision of the service implementation specification, incl. mapping of the UML specification to WSDL and functional service properties (e.g. persistency). Decision on access control policies.

Table 2: Mapping of the RM-ODP Viewpoints to ORCHESTRA

5.3 The ORCHESTRA Reference Model

A graphical depiction of the relationships between the viewpoints and their mapping to the ORCHESTRA architectural design process, the implementation and deployment phase is provided in Figure 2. The result is called the ORCHESTRA Reference Model that covers the following phases:

- Analysis phase that leads to the specification of the Enterprise Viewpoint (see section 6)
- Design phase that leads to the specification of the ORCHESTRA Architecture (see section 5.3.1)
- Implementation phase that leads to ORCHESTRA Implementation Specifications (see section 5.3.2) implemented as ORCHESTRA Service Components
- Deployment phase that leads to ORCHESTRA Service Networks (see section 5.3.3).

The iteration cycles that permit to adapt the architecture to changing or refined needs as specified in the enterprise viewpoint are not shown in Figure 2.

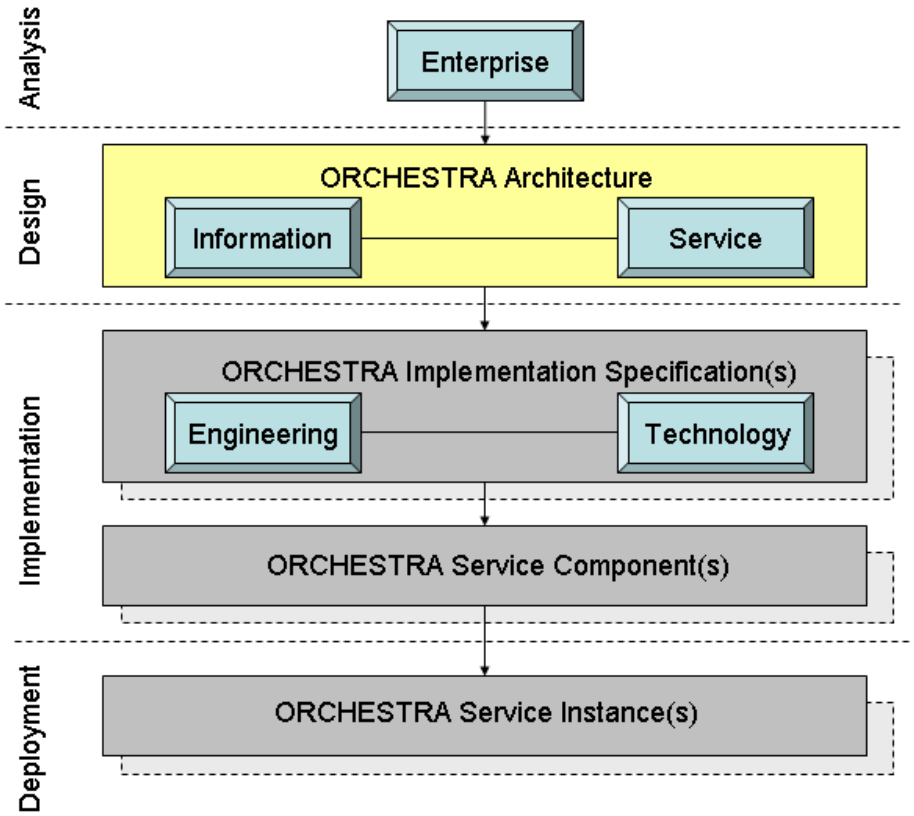

Figure 2: The ORCHESTRA Reference Model

5.3.1 The ORCHESTRA Architecture

The **ORCHESTRA Architecture (OA)** is, by definition, a platform-neutral specification according to the requirements of ISO 19119:2005 (i.e. specification in the conceptual schema language UML). The ORCHESTRA Architecture is specified as part of the design phase. It encompasses the harmonised specification of the Information and Service Viewpoint resulting from requirements of the Enterprise viewpoint. The fact that the specification is platform-neutral means that the ORCHESTRA Architecture does not contain any particular dependencies on the peculiarities of a given platform.

5.3.2 The ORCHESTRA Implementation Specification

The aspects of the Engineering and Technology viewpoints are outside the scope of the ORCHESTRA Architecture. Instead, they are combined in a dedicated specification step that may be carried out multiple times. Each step represents one mapping of the ORCHESTRA Architecture (i.e. the Information and Service Viewpoint specification) to a specific service platform and leads to a platform-specific **ORCHESTRA Implementation Specification (OIS)**.

A **service platform**, or **platform** for short, hereby is defined to be the set of infrastructural means and rules that describe how to specify service interfaces and related information and how to invoke services in a distributed system. Thus, a platform provides a service infrastructure and associated rules for the specification, discovery, composition and invocation of services in a distributed system. Examples of platforms are Web Services according to the W3C specifications or a CORBA-based infrastructure according to the OMG specifications.

An OIS contains platform-specific specifications of ORCHESTRA information models and ORCHESTRA services. This means in concrete terms that the information models expressed in UML have to be mapped to a schema language (e.g. XML/GML or EXPRESS) that fits to the selected platform. Likewise, the abstract specifications of the ORCHESTRA service interfaces expressed in UML have to be mapped to a service description language (e.g. WSDL) that fits to this platform, too. These mapping processes may be done manually or performed (semi-)automatically by a tool.

Note: The iterative design process of the ORCHESTRA Architecture allows designers to re-apply changes in the viewpoint specifications if problems during an OIS specification process occur.

Note that an OIS itself is not part of the RM-OA specification. The RM-OA just provides the architectural framework for an OIS. As a consequence,

- the RM-OA description of the Technology Viewpoint (see section 10) contains guidelines, requirements and rules what has to be considered when specifying a platform, and
- the RM-OA description of the Engineering Viewpoint (see section 11) contains guidelines, requirements and rules what has to be considered when mapping to a chosen platform and in the specification of an OIS. Furthermore, the Engineering Viewpoint also covers engineering principles and guidelines how to design ORCHESTRA Service Networks (see section 5.3.3) and discusses their characteristics.

The implementation phase encompasses the edition of the ORCHESTRA Implementation Specifications and their implementation in **ORCHESTRA Service Components (OSC)** and platform-specific encodings of the information models. An OSC is a component that provides an external interface of an ORCHESTRA Service according to an OIS. Note that the platform-specific encodings of the information models are accessed by means of ORCHESTRA Services, thus they are not explicitly illustrated in the ORCHESTRA Reference Model in Figure 2.

5.3.3 The ORCHESTRA Service Network and ORCHESTRA Applications

An executing manifestation of an OSC is an **ORCHESTRA Service Instance (OSI)**. The deployment phase encompasses the deployment of OSIs on hardware (see Figure 3).

The set of ORCHESTRA Service Instances connected through a communication network is called an **ORCHESTRA Service Network (OSN)**. An OSN thus comprises the set of networked hardware components and ORCHESTRA Service Instances that interact in order to serve the objectives of ORCHESTRA Applications.

Note that the grouping of OSIs into software components and their distribution and deployment on hardware components (e.g. server machines) is not relevant from when specifying the ORCHESTRA Information and Service Viewpoint. The basic unit of an OSN for the provision of functions are the OSIs. One of several OSIs may be deployed as part of one software component.

On a next higher level, software components distributed in a network are grouped together to form **ORCHESTRA Applications**. A software component as part of an ORCHESTRA Application may contain one or more OSIs but, in addition, also other functionality, e.g. functions to built service request messages or to consume response messages.

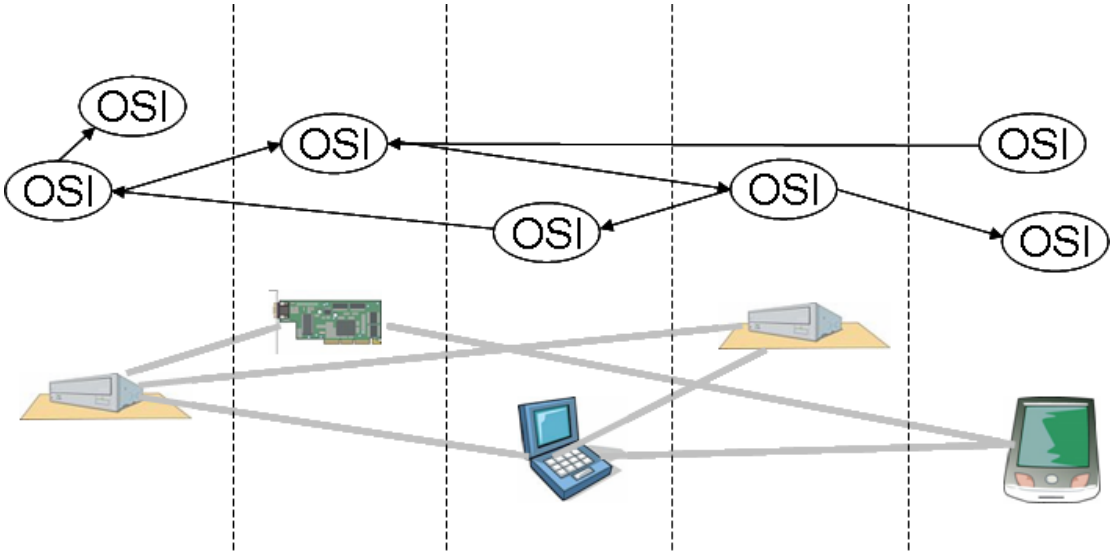

Figure 3: Deployment of ORCHESTRA Service Instance in an ORCHESTRA Service Network

Figure 4 shows the example of two ORCHESTRA Applications that are built out of several interacting software components, some of them containing an OSI and some not. Note that in this example these two ORCHESTRA Applications are sharing the usage of one OSI, i.e., client software components in the respective ORCHESTRA Applications may call operations of this OSI in parallel.

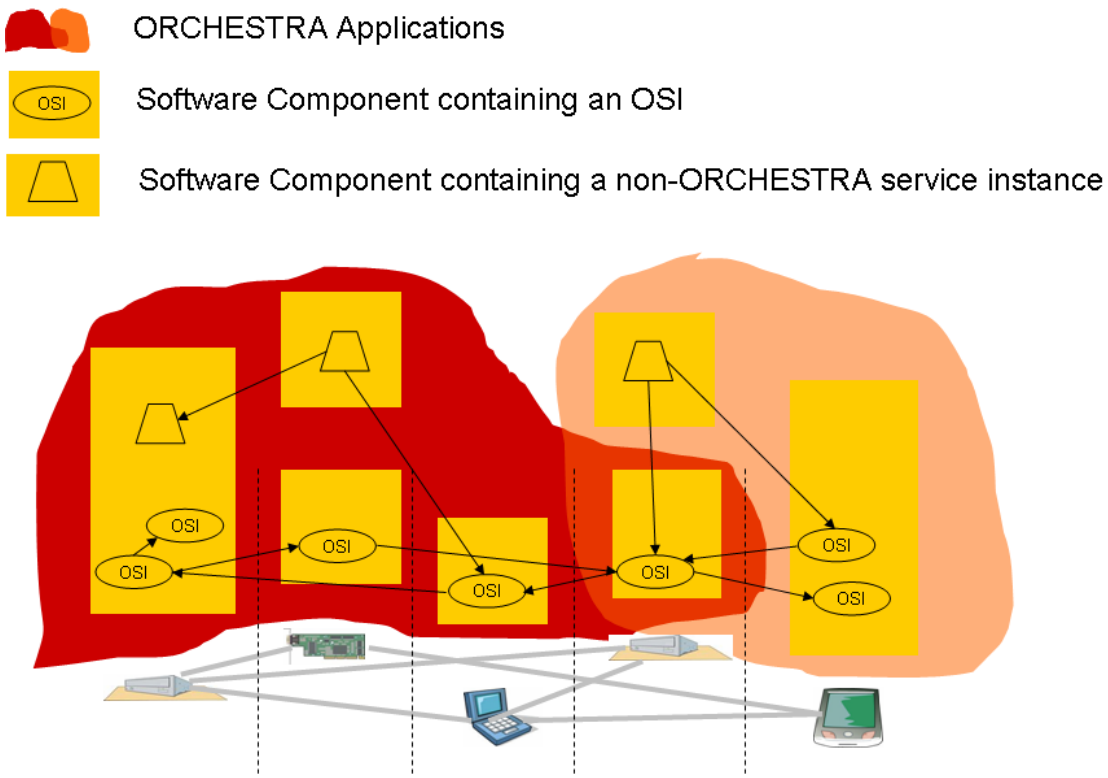

Figure 4: Example of two ORCHESTRA Applications using the same OSI

5.3.4 The ORCHESTRA Application Architecture

An **ORCHESTRA Application Architecture (OAA)** is an instantiation of the ORCHESTRA Architecture by inclusion of those thematic aspects that fulfil the purpose and objectives of a given application. The concepts for such an application stem from a particular application domain (e.g. a risk management application).

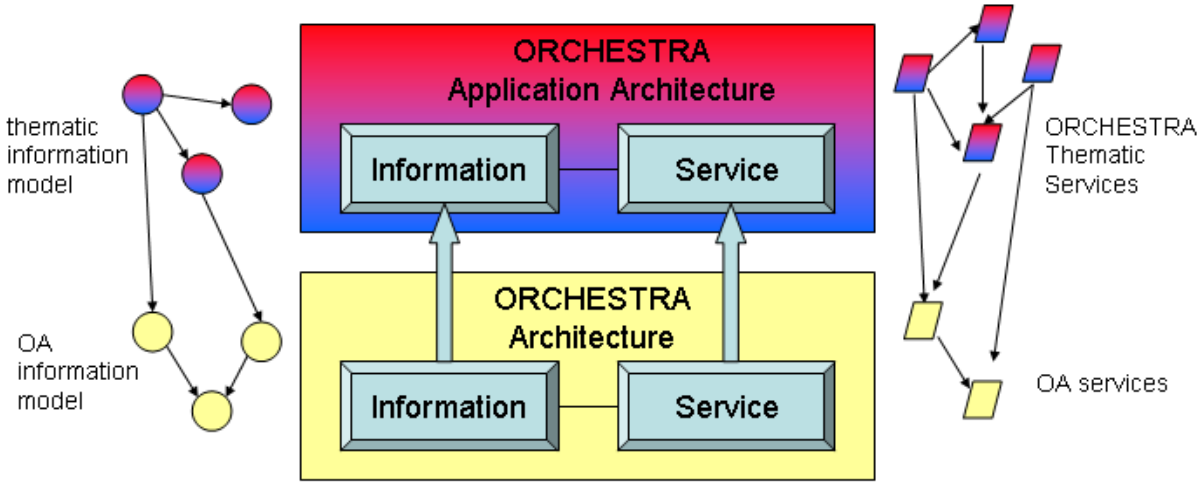

Figure 5: ORCHESTRA Application Architecture

By definition, an OAA is a platform-neutral specification. As such, an OAA covers both the platform-neutral specification of the thematic aspects of the information viewpoint (thematic information model, e.g. a domain-specific ontology) and the service viewpoint (addition of thematic services). It may encompass a specification extension but also a restriction, e.g. omission of optional services or information elements.

The relationship between an ORCHESTRA Application Architecture and the ORCHESTRA Architecture is shown in Figure 5.

Note: The process to identify on the conceptual level the pre-eminent information types and their relationships (leading to a conceptual thematic information model) and the functional requirements (leading to service descriptions on the conceptual level) is outside the scope of the RM-OA. The RM-OA just provides the framework to formally specify information models as well as services in order to integrate them into the OA.

5.3.5 The ORCHESTRA Application Implementation Specification

A platform-neutral specification of an OAA based on a conceptual schema language like UML might not be adequate in all development projects. Sometimes, the platform has been pre-selected and the delivery of a platform-neutral specification that abstracts from the platform specific characteristics is not necessary.

Nevertheless, in order to allow the exploitation and usage of the ORCHESTRA Architecture, the thematic information model and the respective OT Services may also be specified directly on the basis of a chosen ORCHESTRA Implementation Specification. In this case, the resulting platform-specific specification of the thematic extensions and restrictions is called an **ORCHESTRA Application Implementation Specification (OAIS)**.

5.4 The OpenGIS Service Architecture

Topic 12 of the OpenGIS Abstract Specification ("The OpenGIS Service Architecture" - ISO 19119:2005) provides a specification framework for developers to create software that enables users to access and process geographic data from a variety of sources across a generic computing interface within an open IT environment.

It extends the architectural reference model of ISO 19101:2001 defining an Extended Open Systems Environment (EOSE) model for geographic services.

The resulting ISO Architecture for Geospatial Services distinguishes between Information Technology Services (IT services) and Geospatial Information Services (GI services).

- IT Services are general services in a distributed computing environment, such as processing services that perform large-scale computation involving substantial amount of data, system management services for encoding and transfer of data across communication networks etc.
- GI Services are specialized IT services that define capabilities that are specific to the access to, analysis of, transformation of, manipulation of, storage of, or exchange of geographic information.

In the ISO Architecture for Geospatial Services, a GI service is only specified wherever existing IT services of the selected distributed computing platform do not exist or do not meet the specific GI requirement.

In the ORCHESTRA Reference Model the distributed computing platform is referred to as the service infrastructure. However, the distinction between IT and GI services is not applied for the ORCHESTRA service taxonomy because the ORCHESTRA Architecture (and thus the ORCHESTRA services) shall contain an integrated information model that covers thematic, temporal and spatial aspects.

The link of the RM-OA to the technical content of ISO 19119:2005 focuses on the two following aspects:

- the requirements for platform-neutrality (see section 5.4.1)
- the usage of the service taxonomy (see section 5.4.2), and
- the requirements for a simple service architecture (see section 5.4.3).

5.4.1 Platform-neutral and Platform-specific Service Specification

The ORCHESTRA service specifications as part of the ORCHESTRA Architecture shall comply with the requirements of ISO 19119:2005, section 10.3, for "platform-neutrality".

This means that the following points are considered:

- The ORCHESTRA architectural models shall be described in UML according to the rules and guidelines of ISO/TS 19103 (conceptual schema language), e.g. for the usage of basic UML data types.
- As part of the service viewpoint, ORCHESTRA services shall be defined as "platform-neutral service specifications". They both define static models (objects including the attributes and operations for each object) and dynamic models (capturing the interaction patterns between objects and state modelling).
- As part of the engineering viewpoint, the ORCHESTRA platform-neutral models are mapped to a specific service infrastructure context. The resulting platform-specific service models may be defined in UML or in terms of the platform-specific language (e.g. WSDL). However, it is required to maintain a description of their mapping from the corresponding platform-neutral models. This mapping shall show how the intentions of the platform-neutral specifications are met in the context of the service platform. In order to support interoperability, the reverse mapping back to the concepts in the platform-neutral model must be defined.

5.4.2 Service Taxonomy

The ORCHESTRA Architecture informally classifies the ORCHESTRA services according to the service taxonomy of ISO 19101 (also referred to in ISO 19119:2005, section 8.3). The service categories are:

- **Human interaction services** are services for management of user interfaces, graphics, multimedia, and for presentation of compound documents.
- **Model/Information management services** are services for management of the development, manipulation, and storage of meta-information (including ontology specifications), conceptual schemas, and datasets.
- **Workflow/Task management services** are services for support of specific tasks or work-related activities conducted by humans or software components with a high degree of autonomy (agents). These services support use of resources and development of products involving a sequence of activities or steps that may be conducted by different persons.
- **Processing services** are services that perform computations. These computations might range from the performance of mathematical equations up to large-scale computations involving substantial amounts of data.
- **Communication services** are services for encoding and transfer of data across communications networks.
- **System management services** are services for the management of system components, applications, and networks. These services also include management of user accounts and user access privileges.

Note: The classification of a particular service in a taxonomy is considered as meta-information for the service. According to the ORCHESTRA handling of meta-information (see section 8.4.1), the adequacy of this service taxonomy is therefore to be considered when defining purpose-oriented meta-information for services (see section 8.4.2).

5.4.3 ORCHESTRA as Simple Service Architecture according to ISO 19119:2005

The ORCHESTRA Architecture is a service-oriented architecture. Furthermore, looking at ISO 19119:2005, section 7.6, the ORCHESTRA Architecture aims at observing the characteristics of a “simple service architecture” in all cases where it is applicable and useful. Exceptions shall be identified in an explicit fashion.

A simple service architecture according to ISO 19119 and interpreted in the context of the ORCHESTRA Architecture is a message-based architecture that supports service chaining and considers the following simplifying assumptions:

- Message-operations

  ORCHESTRA operations shall be modelled as messages. A message operation shall consist of a request and response. Requests and responses contain parameters as the payload, which is transferred in uniform manner independent of content. Simple applications are characterized by message exchange patterns such as one-way (or event), and two-way (or synchronous) request response interactions. A service specification should make such simple exchange applications as easy as possible to create and to use.

- Separation of control and data

  A client controlling an ORCHESTRA service may not want the full results of the service. For example, the user may have no need for the potentially voluminous intermediate products in a service chain. Only the final result of a service chain may be needed by the client. Therefore, an interface should separate the control of a service from the access to the data resulting from the service. A client should have the option of receiving just the status of an operation and the data should be accessible separately through a separate operation.

- Stateful vs. stateless service

For simplicity it is desired that an ORCHESTRA service be stateless, i.e., that a service invocation be composed of a single request-response pair with no dependence on past or future interactions. This will not always be possible. For some ORCHESTRA services, preconditions must be set and iteration may be required. Then it will be necessary to model the service with a state diagram having multiple states. Transitions between the states are triggered by operations. The state diagram and associated descriptions will be part of the abstract and of the implementation specification of the interfaces of an ORCHESTRA service (see section 9.2.6).

- Known service type

  All ORCHESTRA service instances are of specific service types and the client may access the service type description prior to calling the service. In the ORCHESTRA Reference Model, a “known service type” is a service type with an externally available description.

  Note: The ORCHESTRA Reference Model does not enforce that the “clients shall contain software for accessing the service type prior to encountering service instances of the type in an implemented architecture” as requested by ISO 19119:2005. Although this is useful and a good start in many applications in order to reduce complexity, the ORCHESTRA Architecture aims at providing services that enables the design of generic application code that is controlled by the availability of service meta-information.

- Adequate hardware

  ORCHESTRA Services are implemented as software components (OSCs) and deployed and executed on hardware hosts. The ORCHESTRA Reference Model assumes that the issues of hardware hosting of the software are transparent to the user. It is assumed that the service has adequate hardware, i.e. hardware assignment is transparent to user.

6 Enterprise Viewpoint

6.1 Overview

The enterprise viewpoint of the ORCHESTRA Architecture briefly describes its

- business perspective,
- purpose (the core mission of the ORCHESTRA Architecture),
- scope (e.g. intended users),
- policies (e.g. standardisation approach, openness)

In terms of the architectural process described in section 5, it reflects the analysis phase in terms of the high-level and the user requirements as well as the technology assessment.

6.2 Business Perspective

6.2.1 Contribution to the ORCHESTRA Goals

The design of the ORCHESTRA Architecture (OA) is triggered by the main goals of the ORCHESTRA project which have been described as:

- To design an open service-oriented architecture for risk management where special attention will be paid to providing a solution for the combination of spatial and non-spatial data and services. The ORCHESTRA Architecture will contribute to the INSPIRE (INSPIRE, 2007) (Dufourmont, Annoni, De Groof 2004) and GMES (GMES 2004) infrastructure and thus will assist and support the needed development of INSPIRE technical specifications and guidelines in the INSPIRE preparatory phase.
- To develop a software infrastructure for enabling risk management services.
- To develop services for various multi-risk management applications based on the architecture.
- To validate the ORCHESTRA Architecture and thematic services in a multi-risk scenario.
- To provide software standards for risk management applications, and to provide additional information about these standards. In particular, the de facto standard of OGC and the de-jure standards of ISO and CEN are expected to be influenced.

6.2.2 Collaboration with European Initiatives and Projects

Furthermore, the ORCHESTRA Architecture is meant to provide substantial input to an information infrastructure (info-structure) in the context of the European INSPIRE (Infrastructure for Spatial Information in Europe) and GMES (Global Monitoring for Environment and Security) initiatives, especially but not exclusively for environmental risk management applications. For this task, ORCHESTRA will cooperate with two other European integrated projects:

- OASIS: Open Advanced System for crisIS management (IST IP 4677 http://www.oasis-fp6.org/)
- WIN: Wide Information Network for Risk Management (IST IP 511481 http://www.win-eu.org)

These projects face in common the task of organising risk management systems that are networked across and between organisations with interoperable capabilities.

6.2.2.1 Common Architectural Principles of ORCHESTRA, OASIS and WIN

In June 2004, the European Commission (DG INFSO) has initiated a series of meetings between major stakeholders of the strategic objective “Improving Risk Management”, (i.e. ORCHESTRA, OASIS and WIN), stakeholders of GMES (in particular ESA) and stakeholders of INSPIRE (in particular JRC). With respect to the relationship between ORCHESTRA, OASIS and WIN common architectural principles of an open info-structure have been discussed and were finalised in a white paper (see also section

6.2.2.4).

OASIS and ORCHESTRA have agreed to work on a common scenario that will combine the needs across different phases of the risk management cycles, including the response phase. The scenario will be developed as a paper study which aims at evaluating the OA in a disaster management context.

6.2.2.2 Requirements of the INSPIRE Relationship

The acronym INSPIRE stands for “Infrastructure for Spatial Information in Europe”. INSPIRE is a European directive establishing the legal framework for setting up and operating an Infrastructure for Spatial Information in Europe. The Directive focuses on spatial data that are held by or on behalf of public authorities. INSPIRE is targeting environmental policies, however, other sectors such as agriculture, transport and energy may benefit, too, once this infrastructure is in place.

The proposal of the INSPIRE directive lays down general rules for the various components of a framework for a European Spatial Data Infrastructure (SDI). Thus it considers rules for metadata to support the discovery and evaluation of spatial data and services; rules to achieve interoperability that allows integration of spatial data of the various themes addressed by INSPIRE; rules for interoperable network services for discovery, viewing, accessing and downloading spatial information; rules for data sharing; necessary coordinating structures; and the development of a European geo-portal to provide a common entry to access all INSPIRE network services.

The INSPIRE Directive was published in the official journal on the 25th April 2007and entered into force on the 15th May 2007 (INSPIRE, 2007).

The INSPIRE Work Programme published in April 2005 identified a step-wise approach for the definition and preparation of detailed Implementing Rules (Dufourmont, Annoni, De Groof 2004). Clearly, such Implementing Rules cannot be developed in isolation but need to take into account what already exists in the Member States as well as the broader international developments in the field of SDI and e-government services. In addition operational experiences, international agreements and protocols that are already in place across various thematic communities need to be considered.

With these considerations in mind, an open call was launched in spring 2005 for the registration of interest by Spatial Data Interest Communities (SDIC) and Legally Mandated Organisations (LMO). LMO represent those organisations at local, regional, and national levels that have a formal legal mandate giving them the responsibility for specific thematic data resources. As part of the open call it was asked to put forward experts and reference material to support the preparation of the Implementing Rules. More than 180 experts have been proposed, including experts supported by the ORCHESTRA project. The INSPIRE Drafting Teams were then established and started operations in October 2005.

In early 2007 the ORCHESTRA consortium registered as SDIC.

The current time-line for the full implementation of INSPIRE envisages that the Directive will be transposed in national legislation by the Member States in 2008-9, and that implementation will take place in the following years.

The technical INSPIRE Implementing Rules shall be based on existing standards and specifications if possible. Thus the existing geographic information standards and specifications from ISO, CEN and OGC serve as input into the drafting of the INSPIRE Implementing Rules. If it turns out that these standards do not cover or cannot fully fulfil requirements formulated in the INSPIRE directive adequate extensions and modifications are proposed and respective feedback into the standardisation bodies will be ensured. The current status of the drafting of the INSPIRE Implementing Rules has been reported on the recent 12th EC GI conference in June 2006[1].

Input of ORCHESTRA into INSPIRE could be expected on the drafting of Implementing Rules for Network Services by providing the RM-OA and the developed ORCHESTRA services specifications as reference materials. The requirements on INSPIRE Network Services are therefore detailed in the following sub-section. Moreover ORCHESTRA could support the drafting of Implementing Rules for INSPIRE Data Specifications[2] by providing the RM-OA and the derived application schemata as reference material.

[1] See http://www.ec-gis.org/Workshops/12ec-gis/presentations

[2] See http://www.ec-gis.org/Workshops/12ec-gis/presentations/Plenary%20room/INSPIRE%20I/portele.pdf

6.2.2.3 Detailed definitions and requirements of INSPIRE Network Services

In the context of INSPIRE Network Services the current INSPIRE proposal distinguishes the following service types:

- discovery services
- upload services
- view services
- download services
- transformation services
- "invoke spatial data services" services

Following the INSPIRE proposal, the Network Services will be available from each Member State leading to a distributed architecture at the European level. They will be accessible via the European Geo-Portal and potentially via the member states' own portals. The definition of appropriate technical specifications requires that considered interface specifications are mature and proved by implementation and operational usage, including performance consideration.

As a first task a more detailed description of these network services is developed. The document on *Detailed definitions on the INSPIRE Network Services* [3] proposes a (technical) understanding of the INSPIRE Network Services and tries to identify related issues. This understanding served as a starting point for the work in INSPIRE Drafting Team on Network Services. The Drafting Team is currently updating the document and adding a description of an INSPIRE (service) reference model that includes the concept of horizontal services for DRM, UAA, and e-commerce aspects[4]. The understanding and detailed description of the INSPIRE network services developed so far is summarised in the following paragraphs.

Discovery Services

Discovery services are to search for *spatial data sets* and *spatial data services* on the basis of the content of the corresponding metadata and to display the content of the metadata. As a minimum the following combination of search criteria shall be implemented:

- keywords,
- classification of spatial data and services,
- spatial data quality and accuracy,
- degree of conformity with the harmonised specifications,
- geographical location,
- conditions applying to the access to and use of spatial data sets and services,
- the public authorities responsible for spatial data sets and services.

The related search and response metadata are defined by the INSPIRE Metadata Drafting Team.

The OpenGIS Specification Catalogue Service Web with the ISO application profile (CS-W 2.0 ISO AP 19115/19119) has been identified by the Network Services Drafting Team as the most relevant specification for INSPIRE discovery services. This specification would make the definition of a related set of query and response properties, query language, and the desired level of discovery (dataset only, or also feature level) necessary. As a candidate standard for service metadata ISO19119 has been identified but is not considered to be as well developed as the ISO19115 standard is for metadata.

Another open issue on discovery services is whether and how to deal with multiple application profiles for discovery services (e.g. the ebRIM Profile for the CS-W) and whether and how to link to service registries as UDDI.

[3] http://inspire.jrc.it/reports/dt/ir_dev_process_network_services.pdf

[4] See http://www.ec-gis.org/Workshops/12ec-gis/presentations/Plenary%20room/INSPIRE%20II/serrano.pdf

Upload Services

Currently the upload services are considered to be functionality closely linked to discovery services allowing for the publishing and updating of metadata sets.

View services

The following specifications have been identified by the Network Services Drafting Team as the most relevant specification for INSPIRE view services:

- ISO 19128 Web Map Service
- Draft CEN TC287 profile of ISO 19128 / WMS 1.3

Download services

For INSPIRE Download services it is proposed to distinguish downloading predefined datasets (for instance FTP for downloading files) and downloading features allowing for an appropriate selection of these features (for instance an OpenGIS Web Feature Services). It is envisioned that INSPIRE download services require close links to e-commerce services and the work on INSPIRE metadata and data specification implementing rules.

Transformation services

Services to support coordinate transformation have been identified as an important and thus prioritised instance of INSPIRE transformation services. Within this context the draft OpenGIS Specification for Web Coordinate Transformation Service (OGC WCTS) has been identified as highly relevant. As for the view services questions were raised about the need for and requirements on a (European) CRS Registry.

As further candidates for INSPIRE transformation services, services for schema transformation and services for generalisation have been discussed. Whether these services are required is still under consideration.

“Invoke spatial data services” services

The INSPIRE drafting team proposed invocation services to be understood as the possibility to orchestrate (“chain”) INSPIRE spatial data services in the sense of distributed geo-processing.

The draft INSPIRE Directive requires "invoke spatial data services" to ensure that spatial data services can be invoked in an INSPIRE way fostering harmonisation and interoperability, be it by a user or by other services or applications. If an INSPIRE service reference model includes constraints and characteristics a spatial data service has to fulfil to be effectively invoked inside a framework and the invocation mechanism is unambiguously defined and detailed in an INSPIRE reference model then it could be envisioned that “invoke spatial data services” service implementing rules concentrate on this reference model and the detailed invocation/activation framework.

For defining INSPIRE invocation services or mechanisms it has been realised that orchestration/chaining of geospatial services is in a very early stage. Here, SOAP, WSDL, UDDI, and BPEL are currently considered as relevant technologies and specifications.

6.2.2.4 Requirements of the GMES Relationship

The overall aim of the Global Monitoring for Environment and Security (GMES) initiative is to support Europe’s goals regarding sustainable development and global governance by providing timely and high quality data, information, and knowledge. Access to information has strategic value in the development of nations and regions. GMES will contribute to Europe’s ability to fulfil its role as a world player. This entails the capacity to have independent access to reliable and timely information on the status and evolution of the Earth’s environment at all scales, from global to regional to local. GMES must also ensure long-term, continuous monitoring on a time-scale of at least decades.

A final report for the GMES initial period (2001-2003) is available (GMES 2004). It proposes a way forward for the GMES period 2004-2008. As part of the strategic requirements specifying how to realise the GMES action plan, this report contains assessments and objectives for Data Integration and Information Management in the GMES service context which could be relevant for ORCHESTRA.

6.2.3 Evolution of the ORCHESTRA Architecture

In order to fulfil the business objectives, especially with respect to the GMES and INSPIRE initiative, the ORCHESTRA Architecture considers from the beginning a multi-step approach:

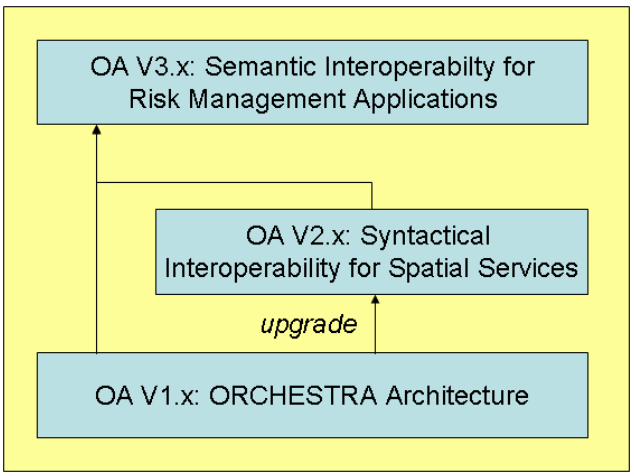

Figure 6: The Evolution of the ORCHESTRA Architecture

- In OA Version 1.x (RM-OA 2005), the ORCHESTRA Architecture has been conceived. The ORCHESTRA Architecture provides a common view of how to harmonise the requirements for syntactic and semantic service and data interoperability including their thematic, temporal and spatial characteristics.
- In OA Versions 2.x (the present RM-OA version), the focus is on refining the OA V1 in terms of service specifications for syntactic interoperability in the spatial domain. These versions link to the INSPIRE requirements for network services as outlined in section 9.4.
- In OA Version 3.x, the focus is on extending and refining former OA versions in terms of service specifications for semantic interoperability in the risk management domain.

Note: None of these OA versions includes ORCHESTRA Implementation Specifications (OIS); they all stay on the platform-neutral level. It has not yet been decided for which OA versions a platform mapping will be provided in the form of corresponding OISs.

6.3 Architectural Requirements for the OSN Design

In the following sections, architectural requirements for the ORCHESTRA Architecture and an OSN are specified. They have been derived through a line of argument starting from

1. the different types of *users* of an OSN and their *roles,*
2. connecting these *user roles* with *fundamental challenges* for the ORCHESTRA Architecture,
3. deriving from that the *key system requirements*, and
4. finally developing *architectural principles*.

Here, only the architectural principles are briefly explained in terms of architectural requirements.

6.3.1 Rigorous Definition and Use of Concepts and Standards

The ORCHESTRA Architecture shall make rigorous use of proven concepts and standards in order to decrease dependence on vendor-specific solutions, to help ensure the openness of the OSN and to support the evolutionary development process of the ORCHESTRA Architecture.

6.3.2 Loosely Coupled Components

The components involved in OSN shall be loosely coupled, where loose coupling implies the use of mediation to permit existing components to be interconnected without changes.

Note: An example of an ORCHESTRA Service Type that supports the concept of mediation is the Catalogue Service (see section 9.7.5) that decouples the resources (data and services) from their clients.

6.3.3 Technology Independence

The ORCHESTRA Architecture shall be independent of technologies, their cycles and their changes. It must be possible to accommodate changes in technology (e.g. the lifecycle of middleware technology) without changing the ORCHESTRA Architecture itself. The ORCHESTRA Architecture shall be independent of specific implementation technologies (e.g. middleware, programming language, operating system) and shall not be influenced by or deal with technical limitations of specific implementation technologies.

Note: The ORCHESTRA Architecture follows this architectural requirement by specifying it in a platform-neutral manner in the first place before mapping it to one or more ORCHESTRA Implementation Specifications (see section 5.3.1).

6.3.4 Evolutionary Development - Design for Change

The ORCHESTRA Architecture and an OSN shall be designed to evolve, i.e. it shall be possible to develop and deploy the system in an evolutionary way. The ORCHESTRA Architecture and an OSN shall be able to cope with changes of user requirements, system requirements, organisational structures, information flows and information types in the source systems.

Note: The iterative design approach in ORCHESTRA resulting in the planned evolution of the RM-OA in several versions (see section 6.2.3) is an example of how this architectural requirement is supported.

6.3.5 Component Architecture Independence

The ORCHESTRA Architecture shall be designed such that an OSN and source systems (i.e. existing information systems and information networks) are architecturally decoupled. This means that the ORCHESTRA Architecture shall not impose any architectural patterns on source systems for the purpose of allowing them to collaborate in an OSN, and no source system shall impose architectural patterns (i.e. service interaction patterns as for instance described in section 9.10) on an OSN .

6.3.6 Generic Infrastructure

The OA Services shall be independent of the application domain. This means that the OA Services should be designed in such a flexible and adaptable way that the OA Services can be used across different thematic domains and in different organisational contexts, and that the update of integrated components (e.g. applications, systems, ontologies) causes little or ideally no changes to the users of the OA Services.

Note: The functional classification of the ORCHESTRA Service Types into application domain-independent and dependent service types (see section 9.3) reflects this architectural requirement.

6.3.7 Self-describing Components

OSN components, such as data elements or services, shall include descriptions of their critical characteristics, including sources, assumptions, etc. The usage of self-describing components that provide context-sensitive formal and semantic descriptions of their interfaces can help to realise semantic interoperability.

Note: An example of how the ORCHESTRA Architecture considers the concept of self-describing components is the mandatory support of the service capabilities interface (see section 9.6.1) that allows a service consumer to learn about the capabilities and the characteristics of a service implementation.

7 Design Decisions of the ORCHESTRA Architecture

The ORCHESTRA Architecture is the combined specification of the ORCHESTRA Information and Service Viewpoints. Both of these viewpoints are specified in dedicated sections (see section 8 for the Information Viewpoint and section 9 for the Service Viewpoint).

However, as concepts introduced in one viewpoint are required for the specification of the other viewpoint, a purely sequential description is not possible. Important design decisions that are not specifically allocated to one of these viewpoints have to be presented in advance. Note that sometimes they are just introduced here but further refined in the respective section. In this case, a forward reference is used.

7.1 Functional Domains of the ORCHESTRA Service Network

The ORCHESTRA Architecture has to face the problem of integrating environmental risk management systems that are networked across and between organisations. It's the OSN, as the running instance of an ORCHESTRA Architecture, that contributes to improve the syntactic and the semantic interoperability of these systems.

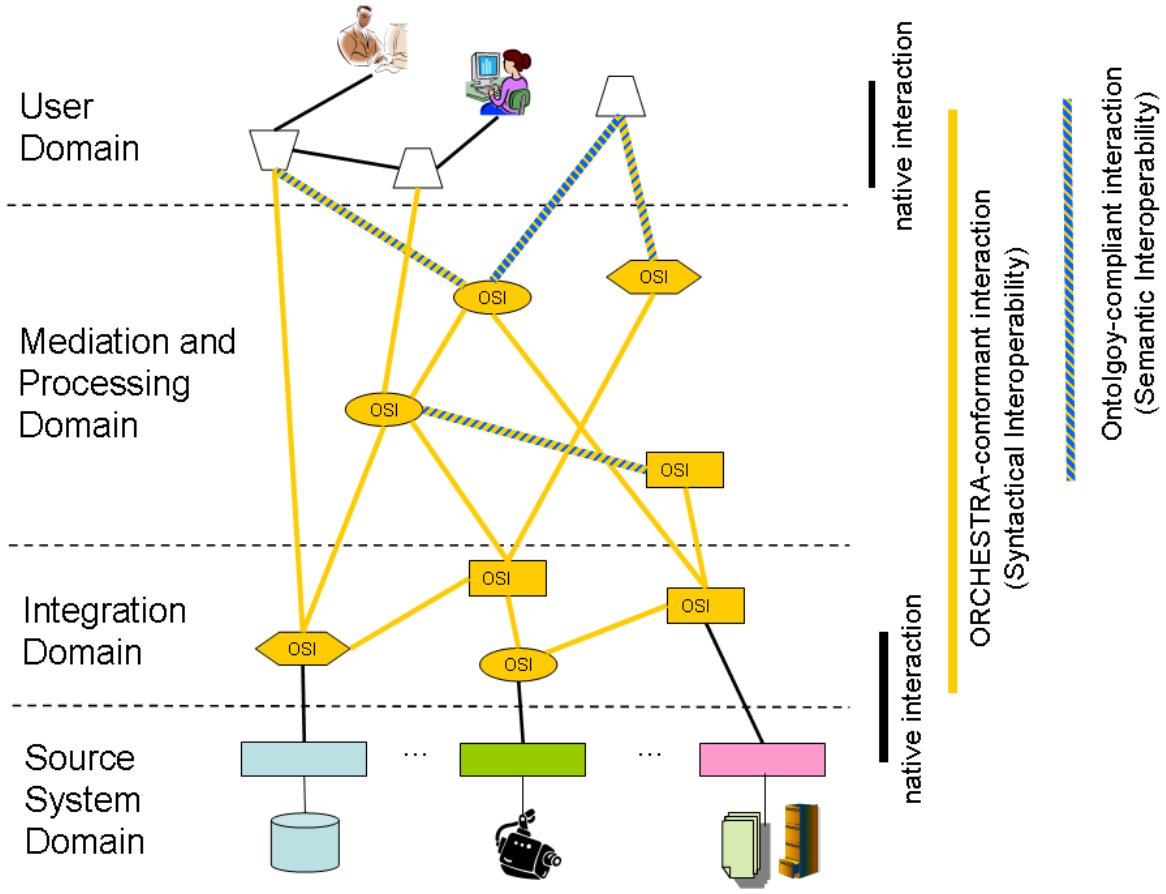

Figure 7: Functional Domains in an ORCHESTRA Service Network

The components of an ORCHESTRA Service Network, i.e. the ORCHESTRA Service Instances (OSIs) are classified according to the following functional domains (see Figure 7):

- User Domain: provides the interface to a user component (a human or a software component) and interacts with the OSIs of the Mediation and Processing Domain according to the rules of the ORCHESTRA Meta-Model. However, user components as part of a (distributed) application may interact among themselves in a native way.
- Mediation and Processing Domain: provides the main functional part of the OSN. It mediates

the service calls from the User Domain to the Integration Domain based on meta-information exchanged with the components of the Integration Domain (e.g. by means of a publishing or a retrieval pattern). Note that the implementation of the services in the Mediation and Processing Domain may be designed themselves as a distributed, possibly functionally-redundant system. The interactions between the OSIs within the Mediation and Processing Domain and with the OSIs in the Integration Domain perform solely according to the rules of the ORCHESTRA Meta-Model. Furthermore, dedicated OSIs in this domain aim at resolving the semantic differences between the information models of the source systems by means of ontologies. Thus, the Mediation and Processing Domain enables semantic interoperability if required by the components of the User Domain.

- Integration Domain: provides support for the source system integration (see below). The OSIs in this domain have two-side interfaces. On the one hand, they interact according to the OMM rules with other OSIs in the Integration Domain and the Mediation and Processing Domain. On the other hand, they interact with the components of the Source System Domain according to their native interface. Thus, the Integration Domain enables syntactic interoperability within an OSN.
- Source System Domain: incorporates the systems and system components of a thematic application area (e.g. risk management) to be coupled. They provide the source of data and functionality and are thus referred to as source systems in the following (see also section 7.6). By means of integration OSIs in the Integration Domains, these source systems are connected to the Mediation and Processing Domain. In practice, this means they need to identify the data and the functionality to be offered in an OSN and to wrap it by respective software components with an ORCHESTRA-compliant service interface. For tightly coupled software systems, this may imply a considerable re-engineering effort.

Note: The platform domain is not visible in Figure 7. It provides the basic communication and encoding mechanisms for the service interactions (the service infrastructure). Its specification is outside the scope of the ORCHESTRA Architecture. The ORCHESTRA Architecture only makes assumptions about the characteristics of the platform (see section 9.2.2.2). Furthermore, in some cases, e.g. in the domain of access control, the platform directly provides support for the implementation of ORCHESTRA Services (see section 10.2).

7.2 The ORCHESTRA Meta-model Approach

7.2.1 Overview

By definition, the ORCHESTRA Architecture shall be generic in the sense that it does not prescribe a specific information model nor an exact configuration of ORCHESTRA Service Instances in an OSN for a given application domain problem. To summarise, the OA is not the specification of a particular information system, but it provides a specification framework for distributed information systems to be used by information modellers and OSN designers. This specification framework provides a set of basic elements to be used and a set of rules to be observed for the purpose of enabling syntactic interoperability between the software components of an ORCHESTRA Application.

These rules and basic elements are summarised in the so-called ORCHESTRA Meta-model (OMM). The OMM consists of two parts:

- The ORCHESTRA Information Meta-model (OMM-Information) that is specified as part of the Information Viewpoint in section 8.7. OMM-Information provides rules about how to specify the application schemas for information models and meta-information models and prescribes the usage of data types.
- The ORCHESTRA Service Meta-model (OMM-Service) that is specified as part of the Service Viewpoint in section 9.2. The OMM-Service provides rules about how to specify interfaces and ORCHESTRA Services and proposes a set of architectural services to be used in an OSN.

Both parts of the OMM are interrelated and depend on each other:

- On the one hand, the structure of the input and output parameters of interface operations have to obey the rules of the OMM-Info.

- On the other hand, built-in operations on feature types have to obey the rules of the OMM-Service.

Note: For convenience, if there is no need to explicitly distinguish between OMM-Information and OMM-Service, the RM-OA simply uses the term OMM to refer to the respective meta-model.

The OMM serves as the basis for checking the conformance of information models and service specifications with respect to the RM-OA. Thus, it has to be specified in detail in a formal and unambiguous way. For convenience, as the OMM is defined in a very formal way as part of the Information and Service Viewpoint the major characteristics are summarised in an informal manner in the following sub-sections.

7.2.2 Major Characteristics of the ORCHESTRA Information Meta-model

In the context of an OSN, information models are specified in order to yield a structure for the information that is potentially being exchanged with an ORCHESTRA Service, i.e. they comprise the structure of the service parameters. The role of the OMM-Information is thus to deliver rules for the specification of such information models (called ORCHESTRA Application Schemas, OAS) with the aim of achieving a harmonised approach for all service specifications and therefore contributing to improved re-usability and interoperability of software components.

The OMM-Information is basically an extension of the General Feature Model (GFM) as defined in ISO 19109. The OMM mandates the usage of UML 2.0 as conceptual schema language.

The central concept in the OMM-Information is that the feature is the basic informational unit as perceived by ORCHESTRA Applications. OMM-Information is a meta-model for feature types. A feature is an abstraction of a real world phenomenon whereby the “real world” explicitly includes hypothetical worlds. Individual features (or feature instances) are grouped into feature types where all instances of a certain type are described by common characteristics.

A feature type contains a set of properties which may be either attributes, operations or associations with other feature types. Furthermore, feature types may be refined by means of inheritance.

The OMM-Information provides rules for the usage of the value domains of attribute type definitions. First of all, for all attribute types it defines a list of basic data types to be used (mostly based on ISO/TS 19103). However, attribute types are further classified into temporal, spatial, location and thematic attribute types with the obligation to use the respective ISO standard definitions (e.g. ISO 19107 and ISO 19125-1 for spatial attribute types).

Attribute types may also represent meta-information about other resources of an OSN. Here, the OMM does not follow the GFM approach of ISO 19109 by strictly requiring the use of ISO 19115. Instead, according to the meta-information approach of ORCHESTRA (see section 7.4), meta-information is always purpose-specific and thus “the” single meta-information model may not be specified. The usage of ISO 19115 in order to define the value domain of meta-information attributes is thus just one of many options.

7.2.3 Major Characteristics of the ORCHESTRA Service Meta-model

The basic structural unit in the ORCHESTRA Architecture as a service-oriented architecture and in an OSN is, of course, the concept of an ORCHESTRA Service. Thus, service modelling plays the predominant role in the specification phase. According to the ORCHESTRA Reference Model, an ORCHESTRA Service is being specified as an ORCHESTRA Service Type, implemented as an ORCHESTRA Service Component (OSC) and executed as an ORCHESTRA Service Instance (OSI).

The OMM-Service provides a meta-model and associated rules for the specification of ORCHESTRA Service Types. Particular emphasis is given to the approach that service modelling is not tied to a particular platform but shall take place on a platform-neutral level (abstract level). The abstraction from platform details improves the mapping from functional user requirements, favours re-use of service specifications for different platforms and enables cross-platform interoperability.

On the abstract level, the purpose and the basic functionality of ORCHESTRA Service Types as seen by the service consumer is described in an abstract description that should be human-readable. The RM-OA proposes the service description framework as introduced in section 9.4 and used later on in the RM-OA for this part. However, there is no formal specification of ORCHESTRA Service Types.

Instead, the OMM-Service defines an ORCHESTRA Service Type as a collection of interface types which specify the externally visible behaviour of an ORCHESTRA Service Type. The concept of an interface type aims at aggregating coherent functionality for a particular objective (e.g. rendering of geographic information in a map) such that it may be re-used for other service types. Thus, on the abstract level an interface type is the unit for re-usability. An interface type itself comprises a set of operations which are the individual units of interaction between a service provider and and service consumer. It is specified in an abstract interface specification and uses UML 2.0 as the conceptual schema language. The OMM-Service proposes dedicated stereotypes for UML classes in order to customise UML for this modelling approach.

An operation is specified by its signature, i.e. its name and its request and response (result and exception) parameters. Here the link between the OMM-Service and the OMM-Information becomes visible: The types of the request and response parameters shall be structured as an ORCHESTRA Application Schema (OAS) according to the rules of the OMM-Information. A parameter value may thus be a value of a basic data type (e.g. an integer) but also a collection of feature instances with their attribute values.

On the platform-specific level, an ORCHESTRA Service Type is represented in an implementation specification that is tailored to the needs and capabilities of a given platform. A selected platform shall be specified beforehand in a platform specification.

Derived from the architectural requirement of “rigorous use of standards” (see section 6.3.1) the OMM-Service assumes that the platform properties and especially the conformance guidelines as specified in the OASIS Reference Model for Service Oriented Architecture (SOA-RM 2006) are fulfilled. As an example, the SOA-RM mandates that the SOA approach of a given platform shall describe how visibility is established between service providers and consumers whereby visibility is understood as follows:

- The initiator in a service interaction shall be aware of the other parties (awareness), e.g. effected by means of a discovery mechanism.

  Note: This aspect is supported in terms of the Catalogue Service described in section 9.7.5 that shall be available at least in all OSNs that are classified as “mediated” (see section 11.2).

- The participants shall be predisposed to interaction (willingness), e.g. a service provider shall respond to an interaction request of a service consumer (except in cases of a denial-of-service attack).

- The participants shall be able to interact (reachability), e.g. it shall be possible to establish a communication path between the participants.

  Note: This aspect is supported by the means for OSN administration. See the Service Monitoring Service as described in section 9.7.10 that shall be available at least in all OSNs classified as „managed” (see section 11.2).

Such a platform specification shall include a description of the principal way in which the mapping from the abstract level is performed (e.g. how an operation is represented), how synchronous and asynchronous interactions specified on the abstract level are principally implemented and how an OAS is mapped from and to UML to the information model langauge of the platform.

For each service type, the OMM-Service mandates that service mapping from the abstract to the platform-specific level is to be specified as part of the implementation specification. The main rules that control the service specification and the mapping are:

- There may be several implementation specifications for one ORCHESTRA Service Type as the implementation specification is platform-specific. However, the OMM-Service also allows several implementation specifications for the same platform by introducing the concept of a service profile (see below).

- Interface types are not visible on the platform-specific level. Instead, their operations are individually mapped upon the action model (SOA-RM term characterising the permissible actions that may be invoked against a service) of the service.

- All ORCHESTRA Service Types shall support the operations of the interface type *ServiceCapabilities* that provide the means to access the meta-information that is associated with a service (e.g. the supported service type, information about the service provider, the set of

implemented operations). A recommendation for a capabilities schema is provided in Annex B1 “RM-OA rules for OAS-MI” of the RM-OA.

- Operations and operation parameters that are marked as optional in the respective abstract interface specifications may be omitted in the mapping to implementation specifications. Thus, service profiles of ORCHESTRA Service Types may be defined, even for the same platform. Their action model thus provides a functional subset of the ORCHESTRA Service Type which is, however, syntactically and semantically compatible such that generic service consumers (application components) may be realised by knowing only the interface types of ORCHESTRA Service Types and the particular platform characteristics.
- ORCHESTRA Service Types that are classified as OA Services (see below) shall first be specified on abstract level and then mapped to the platform level. For all other service types, even if just specified in a platform-specific implementation specification, at least an abstract description of their basic functionality shall be given.

As a consequence of this approach, a community that applies the OMM to specify their services shall maintain a well-defined list of ORCHESTRA Service Types that is consistent between the abstract level and all supported platforms. The RM-OA incorporates as part of its Service Viewpoint in section 9.4 a description of service types that are derived from functional user requirements. This list is further structured into architectural service types (so-called OA Services) that are application-domain independent but indispensable for the operation of an OSN and thematic services (so-called OT Services) that are tailored towards a given application domain. The RM-OA, being a reference model for an application-independent architecture, just provides descriptions of OT Services that support thematic applications across several domains (so-called OT Support Services). Domain-specific services are outside the scope of the RM-OA.

Specifications of the abstract interfaces of the selected ORCHESTRA Service Types are delivered in (ORCH-AbstrServ 2007).

7.3 Resources in an OSN and their identification

There are two fundamental resources in an OSN that need dedicated identification schemes:

- ORCHESTRA Service Instances (OSIs) as the basic functional unit, and
- Feature instances as the basic informational unit.

7.3.1 Identification of OSIs

An OSN comprises a set of interacting ORCHESTRA Service Instances (OSIs) running on top of hardware components connected through a network. In order to be able to search for an OSI and call its operations, a unique identifier of an OSI within an OSN is needed. This unique identifier is also referred to as the name of an OSI in the following.

The name of an OSI is a logical name which may be generated automatically, i.e. it may not directly be meaningful to a human user.

In the case of a dynamic OSN environment that supports the dynamic assignment of an OSI to several OSNs (i.e. the membership of an OSI to one or several OSNs may change during the lifetime of an OSI) an identifier of an OSI that uses an OSN as namespace is not sufficient. In this case, a globally unique identifier is required in order to avoid renaming of OSIs during their lifetime. This means that different OSIs within the same OSN or within different OSNs shall have different names. The OSI name shall be immutable during the lifetime of the OSI.

A recommendation of a naming policy for OSIs that uses the platform as the namespace for an OSI is described in section 11.3.1, however, the usage of this policy is not obligatory. Other naming policies may be defined. The selection of a naming policy for OSIs is one of the characteristics of an OSN as described in the Engineering Viewpoint of the RM-OA (see section 11.1.2).

Note: It has to be distinguished between:

- name of an OSI
- platform-specific identifier of an OSI (e.g. its URI)

- platform-specific address of an OSI (e.g. its IP-address and port)

The focus here is on OSI names and the mapping between OSI names and their platform-specific identifiers. These tasks are related to an OA Service which is called Name Service and introduced in section 9.7.6. The mapping between platform-specific identifiers and addresses is done by platform-specific mechanisms and is out of scope of this document.

7.3.2 Identification of Features

In the same way as an unambiguous identifier of an OSI is required to refer to that OSI within an OSN, each feature instance needs to be uniquely identifiable within the OSN. This is required in order to enable software components in ORCHESTRA Applications to work with references to feature instances instead of performing a query each time feature information is needed.

Such a feature instance identifier shall be immutable during the lifetime of the feature instance. This means that while the values of attributes of a feature may change over time, the identifier assigned to the feature shall not change.

A proposal of a naming policy for feature instances that uses a Feature Access OSI as namespace is described in section 11.3.2, however, as for OSI names, the usage of this policy is not obligatory and other naming policies for feature instances may be defined.

Note that the naming policy of feature instances has to be distinguished from the semantic identity of two feature instances having different names but possibly representing the same real-world phenomenon.

7.4 Meta-information

The terms data, metadata, meta-data, metainformation, information, meta-information, and meta-information are used in different places in the literature, and on the Web.

While most authors clearly distinguish between “data” and “information”, the terms meta-data and meta-information are often used interchangeably. In ORCHESTRA, the meaning of data is only given by the underlying information model, and certain pieces of data may have very different meanings depending on the information model. When referring to certain data in the context of a meta-information model, the RM-OA is actually referring to the meaning given to this data within a model.

In order to avoid confusion, and to account for the fact that all data may have different meanings, the term meta-information shall be used in all the ORCHESTRA documents whenever a datum is seen in the context of a meta-information model (see the RM-OA Annex A3). The related terms, including “metadata”, “meta-data”, and “metainformation” must not be used in the specification parts of ORCHESTRA documents.

The architectural approach to include meta-information in the OA and in the OMM is provided as part of the Information Viewpoint in section 8.4. The argumentation and the foundation for this approach are given in Annex A3 of the RM-OA. A detailed specification of rules and examples is given in Annex B1 of the RM-OA.

7.5 User Management, Authentication and Authorisation

7.5.1 Overview

The access to resources for both feature and service instances is controlled by authentication and authorisation mechanisms. Access encompasses access from human users but also from software components. This is handled by three services: the User Management Service (see section 9.7.6), the Authentication Service (see section 9.7.9) and the Authorisation Service (see section 9.7.8), together referred to as UAA services in the following. An example of their combined usage is described in an OA pattern in section 9.10.1. The general question how many instances of the UAA services are present in an OSN and how they are configured is discussed in the context of UAA policies in the Engineering Viewpoint in section 11.1.5.

The following section just introduces the basic terms and concepts.

Note: Among the variety of security aspects, only the fundamental challenge of how to control the access to resources in an OSN have been considered in the discussion about architectural requirements so far. The reason for this selection is that access control in an OSN is of primary importance when considering cross-border risk management applications.

7.5.2 User Management based on Subjects, Groups and Principals

The major concepts of the ORCHESTRA User Management are subjects and principals.

A **subject** is an abstract representation of a user or a software component in an ORCHESTRA Application. Subject attributes are intended to store generic information about subjects (e.g. first name, last name, address, e-mail, ...).

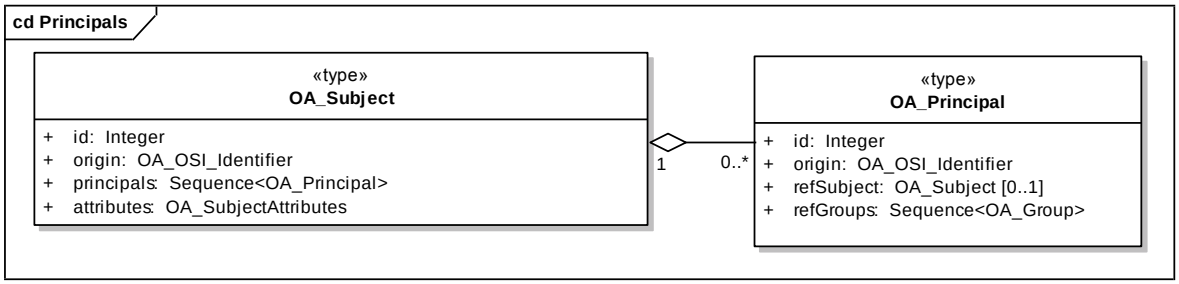

Figure 8: Relationship between Subject and Principal

Subjects need to be authenticated. However, the concept of a subject itself cannot be used for the authentication process. This is mainly because ORCHESTRA aims at supporting multiple authentication paradigms and mechanisms. Their potentially simultaneous use leads to a number of implications, e.g. different authentication mechanisms use different subject representations. Thus, a single subject representation cannot be chosen for ORCHESTRA.

To solve the representation problem, a subject is decoupled from authentication. This decoupling is done by introducing a further concept called a principal. A **principal** is an identity of a subject whereas authentication indicates whether a subject is allowed to use a certain principal. One subject may have multiple principals as illustrated in Figure 8.

Since each authentication mechanism can have its own way of representing a principal, the UAA concept defines a superclass OA_Principal that just contains some attributes used for collaboration purposes (like identifying a principal and referring to the related subject). All attributes that are specific for an authentication mechanism may then be realised by subclasses of OA_Principal.

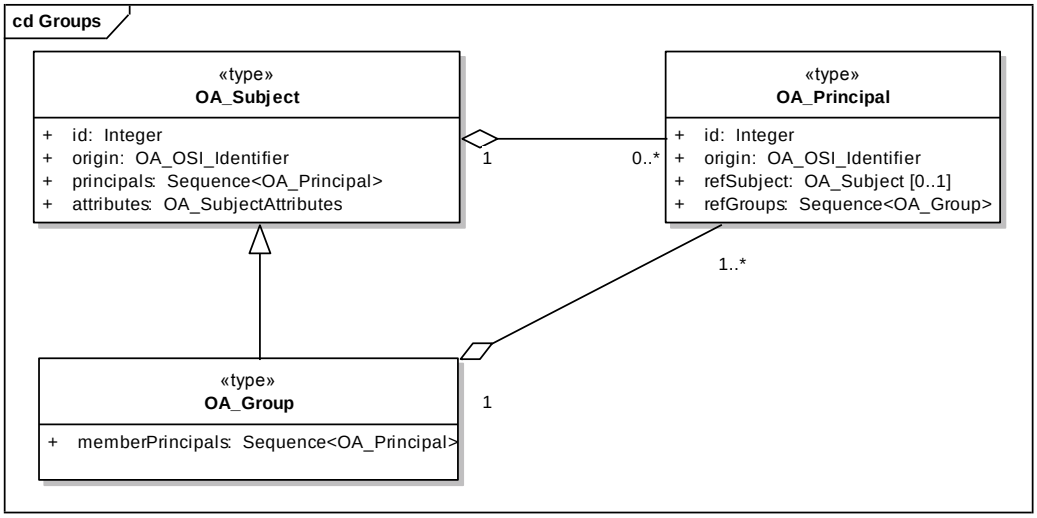

Figure 9: Relationship between Subject, Group and Principal

A **group** is a special subject. A group can have one or more principals (group principals). In addition to principals identifying the group itself a group can have one or more principals as members (member principals). This relationship is illustrated in Figure 9.

Member principals are assigned to a group to define memberships of certain principals.

Based on these concepts, user management is the process of creation and management of subjects, including groups (of principals) as a special kind of subjects. Furthermore, it is up to the User Management Service to associate principals with subjects. The creation and management of principals is up to the Authentication Service.

7.5.3 Authentication

Authentication is the process of verifying the principal of a certain subject. In other words, within an authentication process a subject proves that it is allowed to act with the corresponding principal. Generally speaking, this proof can depend on a secret (credentials) that can be, for example:

- what somebody has (key, smart card, …)
- what somebody knows (password, …)
- what somebody is (biometrical data, …)
- the place somebody resides (certain computer, …)
- the skills of somebody (handmade signature)

The result of an authentication process starts a session that is represented by session information (see section 7.5.5).

Note: As the session information represents the state of the session and must be passed in each service interaction request, it is an example where stateful services are required (see the assumptions of a Simple Service Architecture according to ISO 19119 described in section 5.4.3).

Principals are created and managed in instances of Authentication Services. The process of creating a new principal depends on the authentication mechanism used by the corresponding Authentication Service instance.

After authentication has successfully been passed the Authentication Service generates session information containing the information about which principal has been authenticated.

As an example, consider an OSI of an Authentication Service wrapping an existing Kerberos authentication. Usually a Kerberos implementation ships with a solution for user management. A user in the Kerberos user management becomes an ORCHESTRA principal. This principal then will be associated with the corresponding subject using the ORCHESTRA User Management Service.

7.5.4 Authorisation

Authorisation is the process of determining whether a subject is allowed to have the specified types of access to a particular resource (data or services). This is done by evaluating applicable access control information contained in a so-called authorisation context.

Note: Up to now, only access control for operations has been considered in the RM-OA.

Usually, authorisation is carried out in the context of authentication. Once a subject is authenticated through its principal, it may be authorised to perform different types of access. This is carried out through the concept of permissions that are attached to principals.

A service requests an authorisation decision for a given principal and a given authorisation service context. A service requesting an authorisation decision needs to pass session information containing at least one authenticated principal of the service requestor as well as the authorisation context. Since permissions are bound to principals, the Authorisation Service is able to retrieve permissions for a given principal. There is no restriction on how permissions are associated with principals. This might be done directly or indirectly using roles, for example.

The connection of permissions and principals is essential to the UAA concept by enabling the decoupling of authentication and authorisation. An Authorisation Service may assign permissions to every ORCHESTRA principal, regardless of the mechanism that has been used to authenticate it. This possibility is important. If there is a problem with interoperability – maybe because clients do not support a certain authentication mechanism of a foreign authentication service – they can still use every ORCHESTRA service as long as the corresponding service provider is willing to assign permissions to the client principals.

A group (see Figure 9) can be treated as an ordinary subject by Authorisation Service instances. Thus, assigning permissions to a group does not differ from assigning permissions to any other subject.

Authorisation Services may use different authorisation paradigms. These paradigms can be classified into lookup and expression-based access control.

Lookup based paradigms use predefined data structures to retrieve authorisation decisions. The most famous representative is the role-based paradigm.

Example:

A role-based access control (RBAC) system might use the information model illustrated in Figure 10.

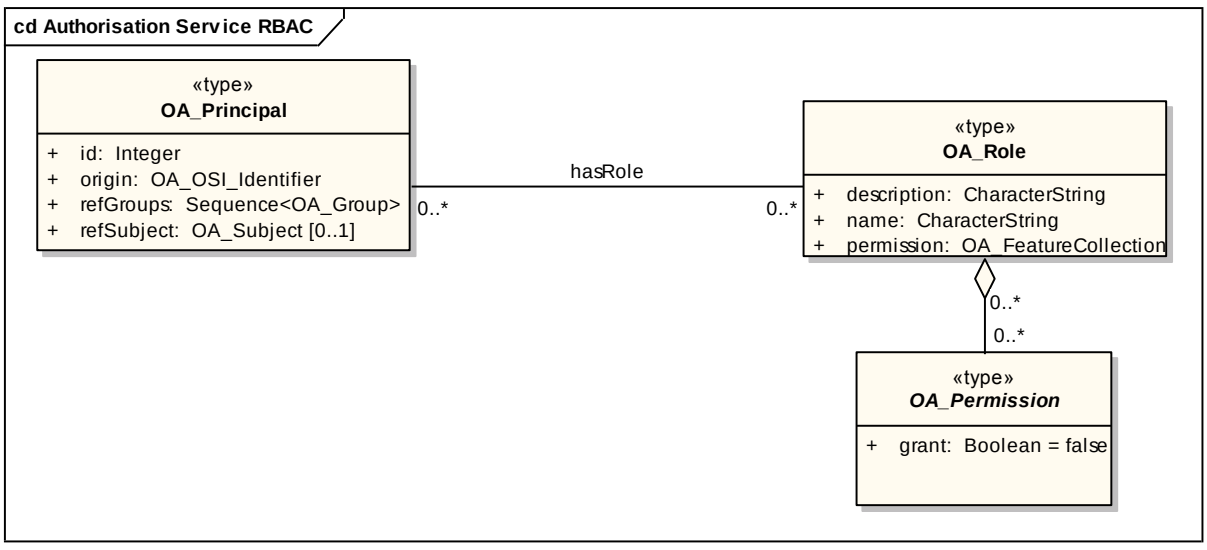

Figure 10: Schema of Role-based Access Control

Expression-based access control systems (EBAC) do not exclusively rely on predefined lookups. More than that, these systems define a framework to specify authorisation conditions. These conditions are parameterised and evaluated in order to compute authorisation decisions. Evaluation of expressions is done by a separated interpreter. This interpreter contains the computational logic and therefore forms

the core of each EBAC.

The most popular representatives of EBAC systems are trust management systems.

7.5.5 Session Information

Session information is created and/or modified by an Authentication Service.

Session information mainly serves as proof that certain principals have been authenticated. Thus, the creation of session information is done by an Authentication Service after successfully authenticating a certain principal.

In order to arrive at an authorisation decision a service needs to know under which principal a service requestor acts. Therefore the requestor of a service has to pass the session information in every interaction with the service instance. Interpretation of the session Information is performed by the invoked service instance.

Verifying and extracting information from session information is a process which is specific to the way session information is treated, e.g. as a session key or as a session envelope. Thus, each service needs to provide a capability, possibly called session handler or session interpreter, which is able to interpret session information as passed from the service requestor.

7.6 Approach to Integration of Source Systems

The OA explicitly takes into account the fact that existing systems and services have to be integrated when designing an OSN. In this respect, it does not matter whether these systems have existed for a long period of time, possibly realised with older technologies, or whether they have been recently designed with modern technology. Thus, the OA uses the term source system to refer to such systems instead of the often-used term legacy system.

A source system is a container of unstructured, semi-structured or structured data and/or a provider of functions in terms of services. The source systems are of a very heterogeneous nature and contain information of a variety of types and and in a variety of formats.

Examples are:

- database containing structured data (e.g. numerical model data), i.e. information that is organised so that it can be easily located, searched, and updated
- database containing semi-structured data (e.g. an XML database)
- database containing unstructured data (e.g. a document archive or image database)
- a system providing services (e.g. a map server)
- Web site, i.e. a provider of a set of html-documents accessible through the W3C http protocol.

For clarification, as illustrated in Figure 11, the OA furthermore distinguishes between an

- External Source System as a source system that does not provide its data and functions through an ORCHESTRA-conformant interface, and
- ORCHESTRA Source System as a source system that provides its data and functions through an ORCHESTRA-conformant interface, in Figure 11 called ORCHESTRA_SourceSystem_IF as an example. This interface shall be built according to the rules as specified in the ORCHESTRA Service Meta-model, in Figure 11 represented by the meta-class OMM_InterfaceType as specified in section 9.2.4.1

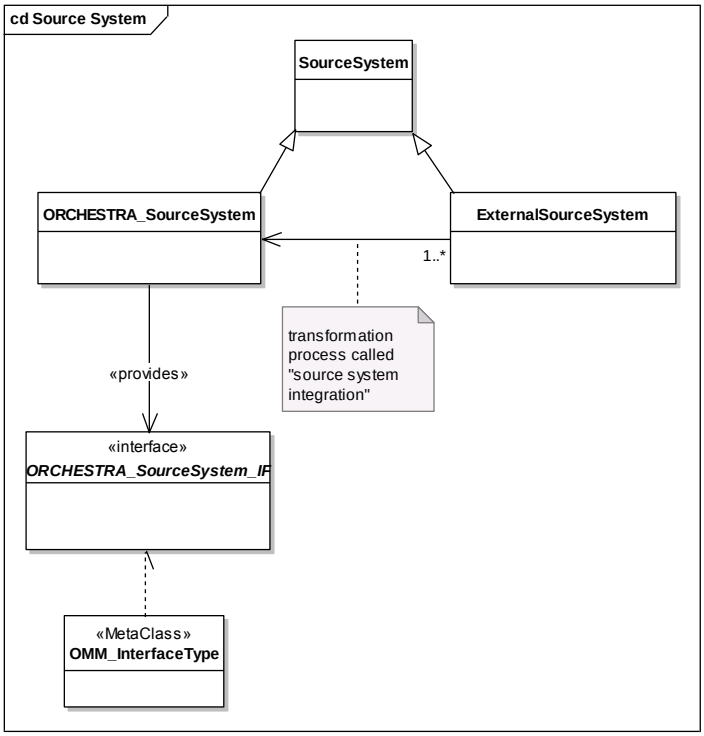

Figure 11: External and ORCHESTRA Source Systems

Each ORCHESTRA Source System is associated with at least one External Source System.

Thus, the major development process for an OSN designer is the process of transforming an External Source System into an ORCHESTRA Source System which is called source system integration.

The OA approach for source system integration is specified in the RM-OA Service Viewpoint in section 9.10.2 as part of the recommended patterns of OA Service usage. The consideration of source systems for the OMM is specified in the RM-OA Information Viewpoint in section 8.5.

7.7 Service Interaction Modes

ORCHESTRA Services will support at least two interaction modes at the conceptual level for the processing of their operations:

- Synchronous mode: In this mode, the requestor principally waits for the response and the response contains the requested data in its output parameters. This mode is usually applied for all operations with a relatively short response time.
- Asynchronous mode: In this mode, the requestor just issues the request for the operation, continues its work in parallel and is asynchronously informed about the availability and a reference to the results. This mode is usually applied for all operations with a longer response time.

Note: These modes are described on the conceptual level which is reflected in respective interaction interfaces of the abstract specification (see sections 9.6.2 and 9.6.3). It does not imply any constraints on the application programming interface in an implementation. This means that a synchronous operation on the conceptual level may be implemented in an asynchronous way and vice versa.

7.8 Interoperability Between Different Service Platforms

Conceptually, there are the following two possible ways to map an OSN onto service platforms:

1. There is exactly one platform assigned to the OSN. In this case, all interactions between all OSIs that participate in the OSN shall follow the rules of this platform (see Figure 12) with the dotted lines representing the logical interaction relationships between the OSIs.

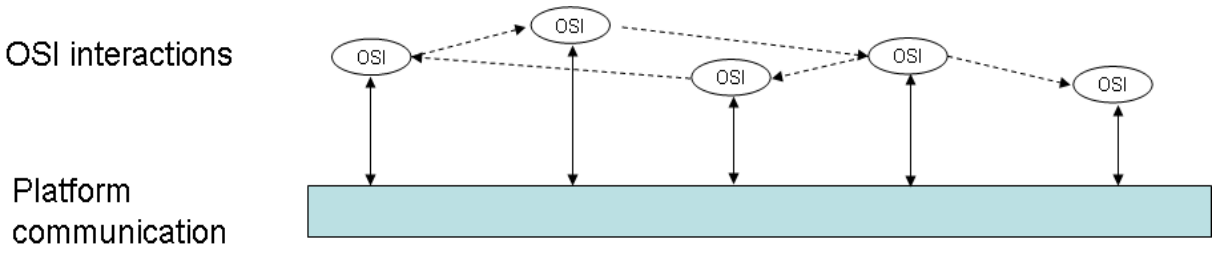

Figure 12: OSI interactions in one platform domain

2. There are several platforms assigned to the OSN sub-dividing the platform into several platform domains. In this case, all interactions between all OSIs that participate in the OSN and belong to the same platform domain shall follow the rules of the respective platform. Furthermore, it must be ensured that all interactions between OSIs that belong to different platform domains are made possible by the provision of respective service platform gateways (see Figure 13). An example for such a situation is a gateway that maps between a CORBA-based platform and W3C Web Services.

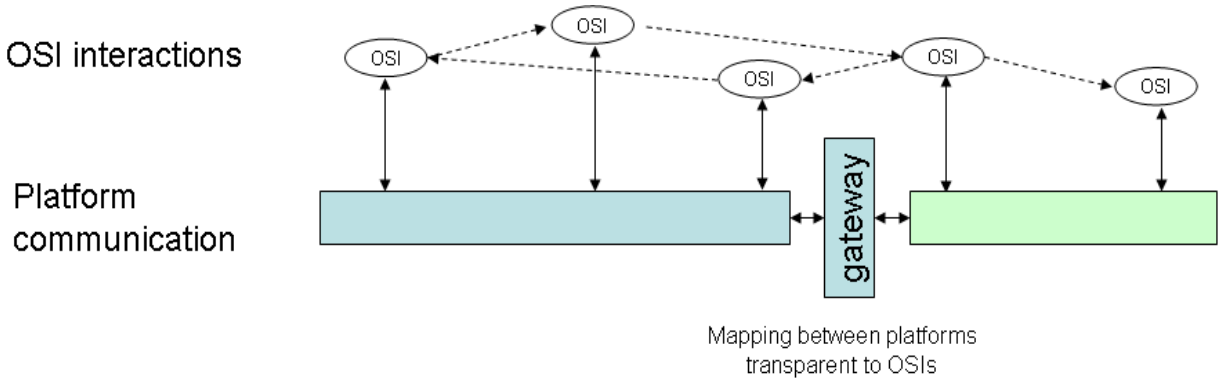

Figure 13: OSI interactions across platform domains

Note: Currently, the RM-OA is restricted to possibility 1, i.e. an OSN may only run on top of one platform that is specified in a given platform specification.

8 Information Viewpoint

8.1 Overview

The Information Viewpoint of the ORCHESTRA Reference Model specifies the modelling approach for all categories of information the OA deals with, including their thematic, spatial and temporal characteristics as well as their meta-information. The ORCHESTRA Reference Model does not specify an information system. Instead it provides a framework for distributed information systems and ORCHESTRA Applications based on a service-oriented architecture. As such, the Information Viewpoint of the ORCHESTRA Reference Model provides an integrated specification framework in order to support a formal specification of conceptual ORCHESTRA information and meta-information models in the context of ORCHESTRA Applications.

This specification framework encompasses the following levels:

- source system level
- feature level
- schema level
- meta-model level
- semantic level

The source system level comprises all the existing systems that contain relevant data or provide relevant services in order to fulfil a particular objective of an application or end-user task (see also the ORCHESTRA functional domains in section 7.1).

The feature level provides an informational view of the data and services of the source system level according to the rules specified for ORCHESTRA features (see section 8.2). Note that no semantic concepts are considered on this level.

The schema level delivers the structuring of information on the feature level in terms of application schemas. Application schemas provide formal specifications of ORCHESTRA Information Models.

The meta-model level provides rules to define application schemas.

The semantic level provides semantics to the information specified in the other levels through explicit consideration of ontologies defined and shared in user communities.

The following sections describe the framework for ORCHESTRA Information Models in two steps:

- In a first step, just the meta-model, the schema and the feature level aspects are considered. For these levels, a specification framework for information models is specified (see section 8.3) and then extended by the consideration of meta-information (see section 8.4).
- In a second step, the specification framework is enriched by considering aspects of the source system level (see section 8.4.4) and the semantic level (see section 8.6).

8.2 The ORCHESTRA Definition of a Feature

One basic concept of the RM-OA Information Viewpoint is the feature, where a feature is an abstraction of a real world phenomenon perceived in the context of an ORCHESTRA Application. A digital representation of the real world can be thought of as a set of features. These individual features (or feature instances) are grouped into feature types where all instances of a certain type are described by common characteristics. The characterisation of features into feature types typically depends on the particular application and is captured in an application schema. This process is shown in Figure 14.

Note: Features have often been understood just as geographic features, i.e. as a feature associated with a location relative to the Earth. The ORCHESTRA definition of features explicitly goes beyond geographic features. It includes tangible objects of the real world but also abstractions, concepts or software artifacts (e.g. documents, software components of IT systems) that may have a physical representation only in software systems. These features may, but need not, have spatial characteristics.

The ORCHESTRA understanding of a “real world” explicitly comprises these hypothetical worlds or worlds of human’s thoughts.

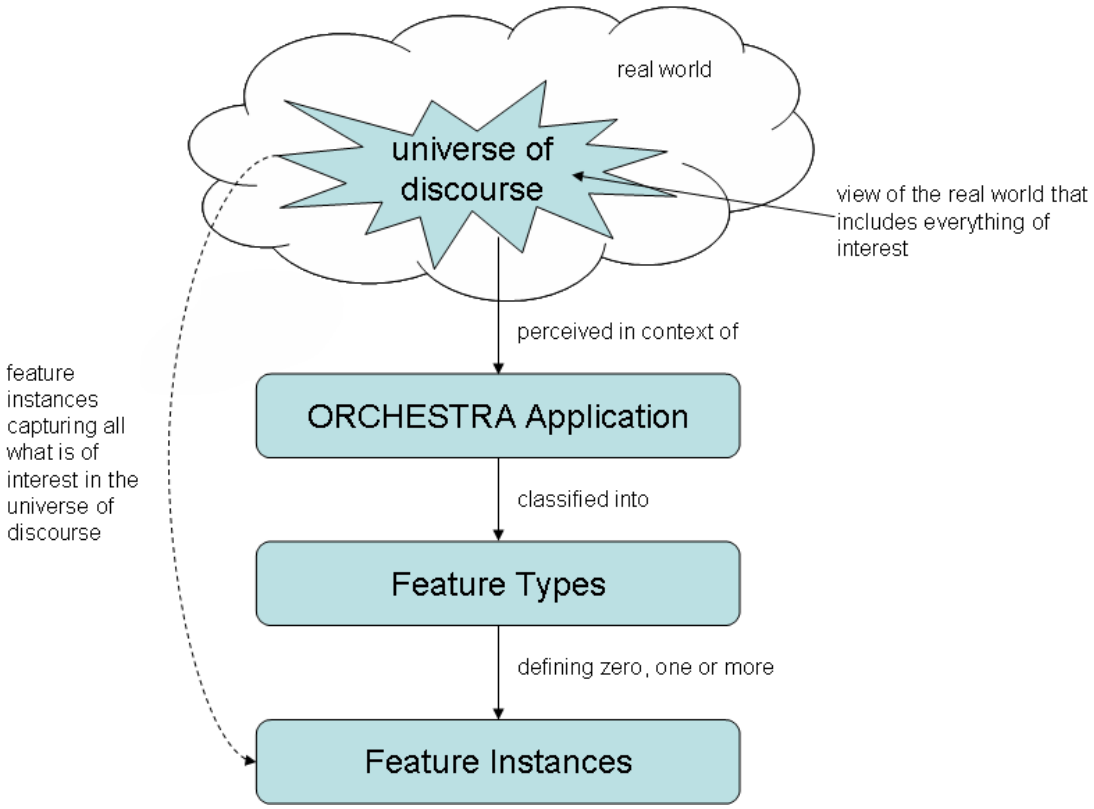

Figure 14: From phenomena to feature instances (derived from ISO 19109)

Common concepts of all application schemas are expressed in the ORCHESTRA feature model as specified in the ORCHESTRA Meta-Model (see section 8.7). Relationships between feature types are feature association types and inheritance. Properties of feature types are feature attributes, feature operations and feature association roles.

Any feature may have a number of such properties. Any feature may have a number of attributes, some of which may be numeric, a spatial geometry, meta-information, temporal information, etc.

Examples of features types are earthquake, forest fire, road, building, water protection area, and monitoring station, but also sensor observation, measurement value, document, and equation.

Examples of feature instances are

- for the feature type “earthquake” the Indian Ocean Tsunami December 26, 2004,
- for the feature type “water protection area” the “Wasserschutzgebiet Seewiesenquellen ID=3463” in the German Federal State of Baden-Württemberg,
- for the feature type “forest fire” the “forest fire near Fréjus in southern France started on July 6, 2005”, or
- for the feature type “document” the “RM-OA Version 1.9 dated July 22, 2005”.

8.3 Framework for ORCHESTRA Information Models

The framework for ORCHESTRA information models distinguishes between

- the ORCHESTRA Meta-Model (OMM) (for information) on the meta-model level,
- ORCHESTRA Application Schemas (OAS) on the schema level and

- ORCHESTRA Feature Sets (OFS) on the feature level.

The OMM specifies the common specification framework for all feature-based application schemas used within ORCHESTRA. It is a meta-model and defines rules for the specification of an OAS. An OAS formally specifies the feature types and their properties which are relevant for a specific information model used in an OSN. It is expressed using the conceptual schema language UML.

The OMM is an evolution of, but it is not a profile of the General Feature Model (GFM) of ISO 19109.

A set of feature instances following the information model formally specified in an OAS is called an ORCHESTRA Feature Set (OFS).

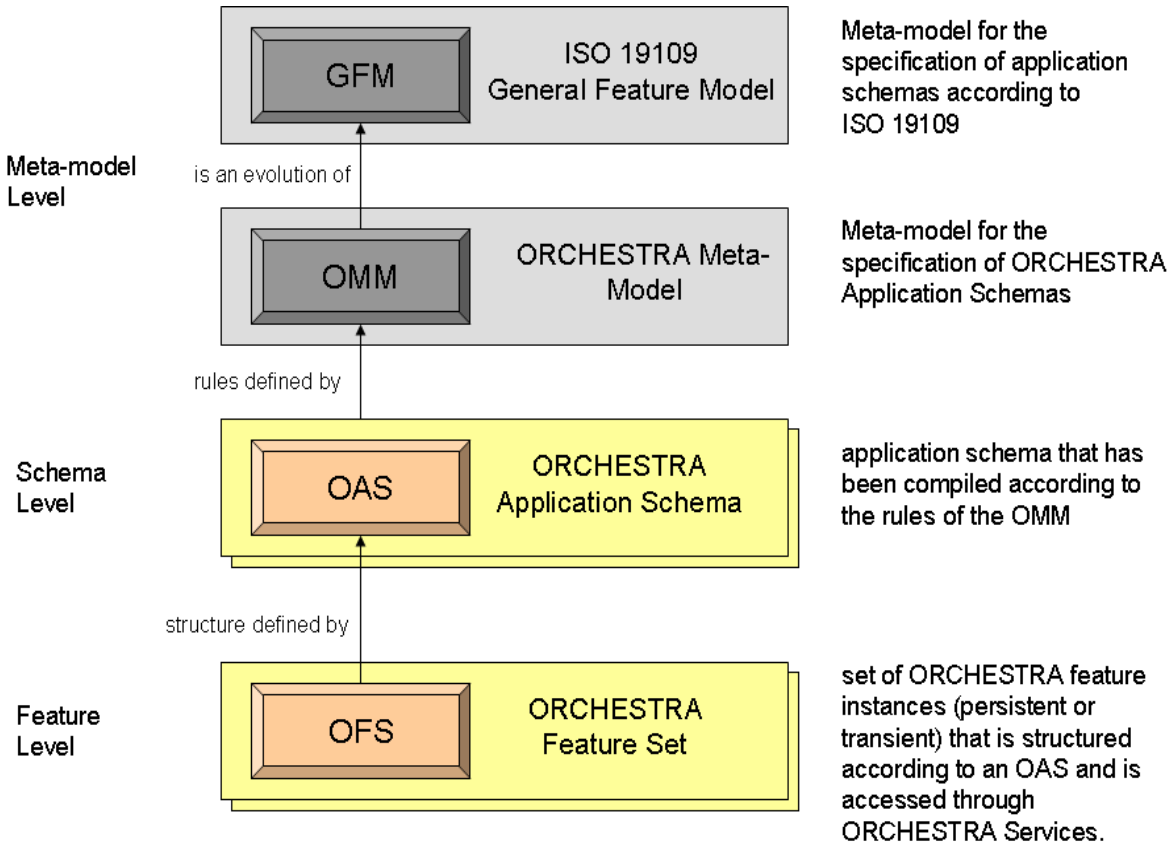

Figure 15: Framework for ORCHESTRA Information Models

8.4 Framework for ORCHESTRA Meta-Information Models

8.4.1 Overview

The following definition for meta-information, which is derived from the principle ideas as described in the Annex A3, is applied for the RM-OA:

Meta-information is descriptive information about resources in the universe of discourse. The structure of the meta-information is given by a meta-information model that depends on a particular purpose. The terms used in this definition are used in the following sense:

- Resources are either functions (possibly provided through services) or data objects.
- Universe of discourse: view of the real or hypothetical world that includes everything of interest (see ISO 19101 and also section 8.2).
- Particular purpose: A purpose of meta-information describes the goal of the usage of the resources. The particular purpose also determines the set of resources in the universe of discourse that are to be considered.
- Meta-information model: a meta-information model represents an implementation of a conceptual model for meta-information. It is represented by an ORCHESTRA Application Schema for Meta-information (OAS-MI).

The above definition indicates that a resource by itself does not necessarily need meta-information. The need for meta-information arises from additional tasks or a particular purpose (like catalogue organisation) where many different resources must be handled by common methods.

Common characteristics of resources in the context of a specific purpose are to be described by means of a meta-information model (concrete by an OAS-MI) that shall be suitable and sufficient in order to define respective algorithms. This means:

1. All information needed to fill up the meta-information model is “meta-information” for this particular purpose.
2. Only attributes of the resources that are also specified in a particular meta-information model are candidates to be meta-information attributes. Specific attributes of the resources that are not specified in a meta-information model are consequently not considered as meta-information for this particular purpose.
3. Meta-information may also be implicitly derived from the existence or content of the resources without requiring that this information be explicitly specified as attribute of the resources. Examples here are the results of annotation services for documents or services that generate meta-information according to a given ontology. This process is known as “classification” in the domain of the Semantic Web.

Thus, the ORCHESTRA Architecture does not define “the” single meta-information model which is valid for any purpose. Instead, in the RM-OA Annex B1, ORCHESTRA defines rules which a meta-information modeller will have to apply to build OAS-MIs related to a dedicated ORCHESTRA Application Schema (information model).

The development process of a meta-information model for data and/or services is guided by the fact that it is necessary to know the purpose of the meta-information. The following approach should be taken:

1. Find the purposes (use cases/functions) in the context of users and/or machines like search, retrieve, etc. (see below).
2. Develop the meta-information model(s) for data and/or services in the respective context.
3. Based on the ORCHESTRA meta-information rules specified in Annex B1 and on the above (step 2) developed meta-information model specify your OAS-MI.

In order to simplify the above process for writing OAS-MIs, Annex B1 offers several example OAS-MIs as a recommendation which can be combined in arbitrary ways to cover a great variety of real world

needs.

The RM-OA defines a set of rules for specifying OAS-MIs for the following “well-known” particular purposes that are further explained in the subsequent sub-sections:

- discovery (including search and navigation)
- access, storage and service invocation
- integration (collaboration, including orchestration and choreography of services)
- interpretation
- user profiling
- authentication, authorisation, and accounting (AAA)
- quality control/management
- transactions, synchronisation and locking
- OSN configuration and management

8.4.2 Description of Purposes

8.4.2.1 Purpose “Discovery”

The purpose “discovery” encompasses methods to find relevant resources within a set of resources, namely search and navigation.

The procedure of searching starts with formulation of a search query that is submitted to the search engine. The search engine returns a number of resources that it has identified as relevant with respect to the query (the search results). Then, the initiator of the query can select resources from the results and/or refine the query.

Examples of meta-information supporting the search procedure are keyword lists, full text index, bounding areas or gazetteer mapping. Examples of services are the Document Access Service and the Gazetteer Service.

Navigation is the process of finding relevant information by browsing within navigational structures. These are provided either by a static or a dynamic catalogue. Examples of meta-information supporting navigation are catalogue entries or catalogue structures; an example of a service is the “Catalogue Service”.

Discovery of services requires a specific meta-information model and dedicated query languages to access the meta-information entries. The type of meta-information needed depends on the quality of the discovery process: discovery might be user driven and based only on syntactic attributes, or it might be automated and based on semantic descriptions.

8.4.2.2 Purpose “Access, Storage and Service Invocation”

The purposes “access” and “storage” are concerned with meta-information needed to access and store data such as exact location information, access protocol, login information, and access rights (see, for example, the authorisation context of the Authorisation Service as described in section 9.7.8). The storage and retrieval will be handled by a “data access service” (in the case of the RM-OA e.g. the Feature Access Service as described in section 9.7.1), so that data access is a specialisation of a service invocation.

Specific meta-information is needed for the purpose of automated “service invocation” based on semantic service descriptions (e.g. OWL-S or WSMO). This requires mapping (also referred to as grounding) of the abstract specifications to concrete service invocation protocols (e.g. SOAP, the protocol for Web Services).

8.4.2.3 Purpose “Integration”

The purpose “integration” comprises aspects of data integration and service integration.

Meta-information for data integration incorporates the description of data, its location, the mappings between different data representations, and data retrieval.

Meta-information for service integration is needed to support composition and interoperability of services. It comprises the description of the service interfaces and functionality.

As an example for an integration requirement, a simulation service based on a flood forecast model and a database containing meteorological data could be imagined. It should be possible to use the database as input for the simulation model and the model’s output as input for any other integrated service.

Service composition is the process of selecting, combining and executing of services in order to achieve a user objective; from the user point of view, the composition is a new service.

A composition is based on a choreography, which defines the rules to communicate with each service participating in the composition in order to consume its functionality. Compositions of services can be distinguished by the time at which the composition is determined: proactive composition (determined at the design phase) and reactive composition (built dynamically at the time the new service is requested). Meta-information is needed for both patterns.

Service interoperability means mutual usage of open service interfaces and protocols across institutional boundaries. However, internal details of the organisation of an institution should not be made publicly visible. Therefore meta-information is required in order to describe the external behaviour of services such that no information about internal business processes is exposed.

Service mediation resolves incompatibilities that arise when performing tasks concerned with the purpose of discovery, invocation or orchestration of services. For instance, in a discovery scenario, queries (formulated by the requestor) and capabilities of services (formulated by the service provider) may be incompatible because they use different terminologies. Incompatibilities can arise on the data level and/or the process level; at the data level, mediation between different terminologies requires solving the problem of ontology integration. At the process level, mediation between heterogeneous communication patterns is necessary in order to resolve possible mismatches, e.g. by generation of dummy acknowledgements.

8.4.2.4 Purpose “Interpretation”

The purpose interpretation is concerned with the support of explanation and understanding of resources (data and services).

In many cases resources can be interpreted only by investigation of vast amounts of implicitly expressed semantics. Thus, explicit descriptions of the semantics shall be added in order to make data and services self-explanatory and enforce their semantic integration.

A real world example is given by a user needing some information about contaminated sites and their classification according to risk categories. Although he has no access to the database containing all the measurements of toxic substances, in some cases he might have to explain the origin of the category number. Therefore he needs the specific measurement values along with the corresponding critical values that caused this classification.

8.4.2.5 Purpose “User profiling”

It is necessary to provide views on data and services and interaction procedures to support different types of users on a per-user or a per-task basis.

Users and tasks will be described in a way that appropriate views on data and services can be provided for different users and tasks.

The required meta-information relates to the way users are represented in an ORCHESTRA Application as subjects (see section 7.5.2). For example meta-information might be user information (user group, service provider, service/data integrator, administrator, etc.) and a particular language.

8.4.2.6 Purpose “OSN Configuration and Management”

Each OSN has to be monitored and administered.

Meta-information for configuration management of the OSN comprises descriptions of the topology of services of the entire OSN, e.g. which services are available at which sites.

Meta-information for the OSN monitoring comprises information on the actual load, service statistics as well as execution traces of services, which are important especially to document and trace execution of services which have been composed reactively.

In order to be able to fulfil this task, all of the services within the OSN have to provide at least their self-description as meta-information.

Means for monitoring, configuration and administration of the OSN have to be provided in order to facilitate this task.

8.4.2.7 Purpose “Authentication, Authorisation, and Accounting (AAA)”

The purpose “accounting, authentication, and authorisation (AAA)” is concerned with meta-information needed for controlling access to computer resources, enforcing policies, auditing usage, and potentially providing the information necessary to bill for services and/or information. Therefore, AAA requires a special set of meta-information that is directly related to the authorisation paradigm and is of little to no use for anything else. This special set of meta-information makes up the authorisation context.

An authorisation context is a set of information used by the Authorisation Service (see section 9.7.8) to determine the authorisation decision for a given request. The authorisation context can contain, for example, the requesting principal(s), name of the invoked operation, etc.

Authentication is a method for identifying the acting subject (representing users or software components in an ORCHESTRA Application) in an OSN. Authentication systems provide answers to the following questions:

- Who is the subject ?
- Is the subject really who he/she purports to be?

Actual mechanisms used for the authentication can be as simple (and insecure) as a plain-text password challenging system or as complicated as the Kerberos system. All authentication systems rely on at least one of these three factors:

- *Something you know*, such as a password or a personal identification number. This assumes that only the owner of the account knows the password or the personal identification number needed to access the account.
- *Something you have*, such as a smart card, a token, or one end of a quantum key generator. This assumes that only the owner of the account has the necessary smart card or token needed to unlock the account, or that he/she is the only person able to access this end of a quantum key generator.
- *Something you are*, such as fingerprint, voice, retina, or iris characteristics.

The ORCHESTRA Architecture does not impose any limitations on the number and type of authentication systems used within OSNs. Unless such limitations are imposed on the implementation level, every service provider in a typical OSN will be free to use its own authentication system.

Typical authentication-related meta-information includes a principal, which is used by the system for authorisation and accounting purposes and therefore should be uniquely assigned to a well-known subject, and some kind of information that is presumably available only to that subject that attempts to authenticate a principal (e.g. ”password“). Independent of the authentication system, at least one centralised or distributed database with user identifiers must exist. In other words at least one OSI of an Authentication Service shall exist in an OSN that is classified as “access-controlled” (see the discussion on OSN characteristics in section 11.1). Depending on the authentication system, this database will also contain shared secrets. Subjects must prove their authenticity by supplying the correct secret. Also, more sophisticated authentication-mechanisms (e.g. one-way hashes of a shared secret, actor’s public key, a list of single-use keys, etc.) taking the place of the “username-password mechanism” are

imaginable.

In security-critical applications, authentication has to take place before granting access to the requested service operations. As a complex (and perhaps extreme) example, an organisation may wish to implement an authentication mechanism involving a retina- and a fingerprint-scan as a pre-requisite for using their PCs, use quantum key encryption over a quantum channel to secure transmission channels and to assure the end-point's identity, restrict access to specific hosts, and finally use some more classical means of authentication before actually granting access to specific service.

Authorisation protects resources by restricting usage of those resources to those principals that have been authorised to use them. The authorisation process is used to decide if that subject is allowed to make use of a specific resource. In order to identify those subjects the authorisation process makes use of the authentication process.

Apart from a static authorisation list the authorisation-decision might also be based on certain dynamic restrictions like time or date constraints, maximum number of concurrent accesses or location-based restrictions (e.g.: no rights granted to remote accessing actors). The types of permission (operation permissions, time-slice permissions) actually supported depend on the implementation of the Authorisation Service (see section 9.7.8).

Authorisation related meta-information may be as simple as a static authorisation list maintained on a central authorisation server, or as complex as a hierarchical set of dynamic rules involving position in an organisation, time or date constraints, maximum number of concurrent accesses or some other measure for service load, billing, or location based restrictions. Authorisation related meta-information is delivered via or referenced within the authorisation context.

The authorisation context is passed to the authorisation service by the service requesting the authorisation decision.

Note: Authentication and authorisation are critical factors for joining OSNs. Whenever two OSNs are joined, a compromise will have to be made concerning the allowed access levels for actors authenticated by the "other" OSN. In the case of the complex example described above, in-house security policy may completely prevent direct merging of "their" OSN with any other network.

Accounting is the process of gathering information about the usage of resources by subjects. This can, for example, include duration of usage or size of the retrieved resources. Accounting information can further be used to support billing, fair-use, planning and many other purposes. In that sense accounting information can be used by the authorisation process in order to provide a basis for the granting of usage rights. The requirements on the actual implementation define the necessary pieces of information and obviously the implemented logic inside the AAA-related and user management services.

Meta-information related to accounting is usually a combination of the principal identifying a subject (e.g. the login-name), and some measure for resources utilisation, such as "amount of data downloaded from the service", "time required to calculate the answer", "duration the resource was used during working hours", "tons of emitted CO_2", or "m^3 of water used for irrigation". Depending on the business model, accounting information may be connected to some kind of a group identifier ("organisation"), or even be completely anonymous.

Note: Due to a lack of user requirements on accounting, dedicated accounting services and meta-information models are currently out of scope of the RM-OA.

8.4.2.8 Purpose "Quality control/management"

The purpose "quality control/management" is concerned with meta-information needed to enhance quality of information and services as well as to increase trust in information, data and services.

Quality control/management is needed when certain criteria need to be fulfilled by data and/or services. Quality usually has different aspects depending on whether services or data are considered. Specifically, quality control is important to every actor in every OSN and highly relevant whenever data and services have to meet certain legal requirements. Therefore working with data that have no quality information may be in some cases just as bad as working with randomly generated data.

Service quality in the ORCHESTRA sense has to deal with infrastructure properties. Examples of these are response time or availability of services. Another aspect that can be considered to be an attribute of

service quality is the fee one has to pay to use the service. Quality regarding the output of services, whether it's back to the actor invoking the service, passed on to another service or stored in an internal data repository is considered to be the data quality of the service. This type of quality is important especially in the context of service chaining when accumulation of errors becomes an issue. A valid source of information for this can be found at (W3C 2003).

Data quality becomes an issue when working with data. Quality may refer to many different aspects and only an open list can be given to characterise them in the context of data:

- absolute and relative errors of measurement data
- computational errors of data processing services
- numerical issues
- minimum and maximum degree of detail in the values of a data set on a specific service
- sensitivity to error accumulation
- refresh period of the data (if it's not just a repository for old data)

Obviously the list of criteria for data quality can become quite long but this degree of detail is not always needed in order to classify the quality of data. The meta-information entries required depend on the particular requirements of the ORCHESTRA Application.

Quality management also means trust management. These are tightly coupled. Trust becomes an issue whenever authenticated and authorized but unknown (or not well-known) parties join a network. When providing their data and services to the network they can and must apply meta-information regarding the quality of what they are exposing. But how can an actor be sure if this meta-information really represents the quality of the actual data and services? The actor's only choice is to either trust or distrust the actor that attached the quality meta-information. Besides deciding whether to trust an actor or not, degrees of trust can also exist. Many different information items can be considered important for trust relationships, including

- Information about the actor: e.g.: name
- Certificates the actor has been granted
- The organisation that the actor represents

Note 1: In order to trust an actor, that actor must be identified first, so a trust relationship relies on the authentication process. Trust relationships are not mandatory but are highly recommended to ensure the quality of a network. A network that does not foresee trust management can be seen as a network where every actor is fully trusted by default.

Note 2: For a discussion on trust in a service environment, see also (OASIS 2006).

Examples for data/information-related quality meta-information: This depends on the data or information item itself. It is important that each of them has attached meta-information. For example a measurement value within an air quality monitoring network can have attached meta-information about its verification status (checked/unchecked) and validation status (valid/invalid).

Examples for service-related quality meta-information: The most important type of service-related quality meta-information is the one concerning guaranteed availability of service and guaranteed response times. For example, a single server has far lower guaranteed availability than a redundant server farm, and a huge grid may be able to guarantee answer times (with constant data quality) practically independent of load. Other important aspects of service-related quality meta-information include "guaranteed availability of the service for next N years", "versioning" (which implies availability of all data for long periods – possibly the whole service lifetime), and "transaction safety".

8.4.2.9 Purpose "Transactions, Synchronization and Locking"

The ORCHESTRA Architecture defines a set of services that are built with interoperability in mind. In order to use the ORCHESTRA Architecture to its full extent, different services need to be transparently combined into new "(virtual) compound services". Using such service chains (combinations) to the full extent requires mechanisms and meta-information that support building transaction-secure composed operations on the OSN level. These mechanisms can be further separated into Transactions, Synchro-

nisation, and Locking.

Transactions are needed when certain tasks that involve resources need to be carried out and it is important to ensure that the resources are not altered during this process.

Synchronisation is needed to secure that data/information are in a consistent state. That means interconnected data have to be kept synchronous.

Therefore, updating distributed data without transactions is dangerous in two ways:

- First, distributed data will inevitably become “out of sync” during the update procedure. Accessing the data while they are still “out of sync”, can lead to unpredictable outcomes.
- Second, the update procedure may break during execution, leaving the data in an unsynchronised state. Consequently, application programmers have to invest a considerable amount of work in checking the data consistency and assuring that the update is eventually completed.

Neither of these problems occurs if all the changes are encapsulated within a single transaction.

A transaction is a logical group of operations that succeeds or fails as a group. This means that either all tasks within a transaction are carried out or none are. That way a transaction appears to be atomic. A lock is a mechanism to (temporarily) restrict the access rights to a resource for certain actors. Locking is used to guarantee the atomicity of transactions.

Note: Care must be taken when using a locking concept in order to avoid deadlocks.

Examples of meta-information related to Transactions, Synchronization and Locking include “start transaction”, “end transaction” and “abandon transaction” signals, and various exceptions signaling that a service is unable to perform a transaction (e.g. transaction unsafe services), had to abandon a transaction because part of it did not work out (e.g. one service in the chain isn’t transaction safe), or that a service is unable to respond to a request because it is currently busy with an transaction.

In addition, each transaction/synchronization request to a transaction safe service produces a lock that is unique with respect to at least this service and thus also unique with respect to OSN (because service has unique identifier with respect to OSN). In order to minimize problems with deadlocks, it may be advisable to assign an OSN-wide unique identifier to each transaction, maintain a globally accessible list of transactions and locks they are causing, and enforce an OSN-wide policy on maximal acceptable transaction times.

8.4.3 Framework Specification

The framework for ORCHESTRA Meta-Information Models (see Figure 16) is specified according to the general considerations for meta-information as described above. It distinguishes between

- an ORCHESTRA Meta-Model (also used for meta-information) on the meta-level,
- ORCHESTRA Application Schemas for Meta-information (OAS-MI) on the schema level, and
- Meta-Information Bases on the feature level.

The Meta-Information Base is a store for meta-information elements. The store might be persistent or transient, depending on the purpose of the meta-information usage. An example of a persistent store is a catalogue for discovery or navigational purposes. An example of a transient store is the usage of meta-information that is extracted on-the-fly in order to support mediation tasks. The Meta-Information Bases contain information that describes features in the form of an OFS according to a well-defined purpose (e.g. navigation, search). There may be several Meta-Information Bases in an OSN.

The structure of these Meta-Information Bases is defined in dedicated ORCHESTRA Application Schemas for Meta-Information (OAS-MI) as a special variant of OAS applied to meta-information. As the Meta-Information Bases are generated according to some purpose, there may be different OAS-MIs for different purposes. ORCHESTRA does not specify one conceptual schema for meta-information models for all tasks. Instead, the ORCHESTRA Meta-Information Model consists of the set of all OAS-MIs that are defined according to the purposes identified above.

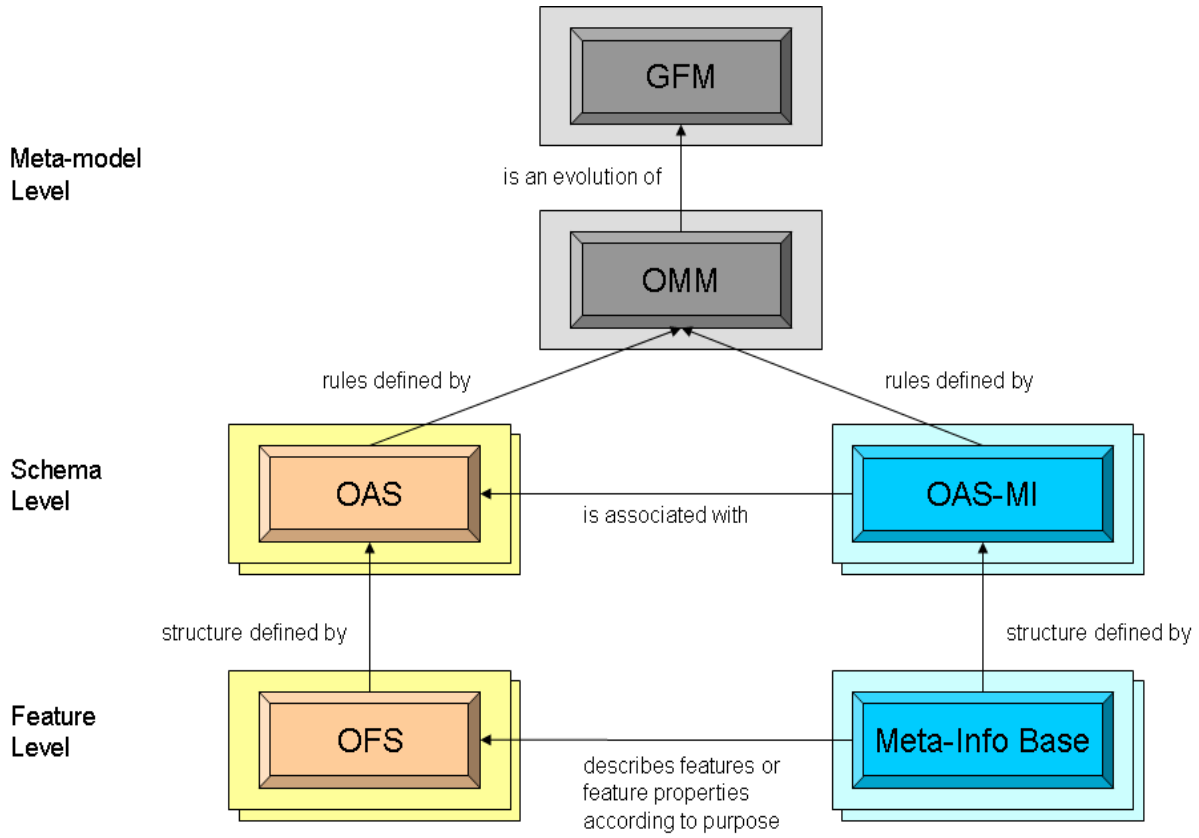

Figure 16: Framework for the ORCHESTRA Meta-Information Model

Depending on the purpose, an OAS-MI may be related to an OAS through some relationships between the two models, e.g. the OAS-MI elements may be attribute types of feature types or they may be feature types themselves that are associated with other feature types.

The meta-model for the OAS-MI is the OMM with dedicated statements on the role of attributes that are considered as meta-information for a particular purpose (see section 8.7.4). Thus, all rules for OAS also apply for OAS-MI.

Dedicated rules for the definition of OAS-MI are defined in Annex B1 of the RM-OA.

8.4.4 OMM Extensions for Meta-information Association Types

In order to allow one OMM_FeatureType instance to serve as meta-information for another OMM_FeatureType instance another subclass, OMM_MetaInfoAssociationType, is added to OMM_AssociationType (see Figure 17). This means that in an OAS, classes marked as feature types can be associated with each other using instances of the OMM_MetaInfoAssociationType.

Note 1: The list of subclasses is not complete in Figure 17 as new or refined classification schemes could be applied, e.g. different variants of aggregation.

Note 2: This approach covers meta-information for Features, Feature Collections and Feature Types as all three terms can be subsumed under the term feature.

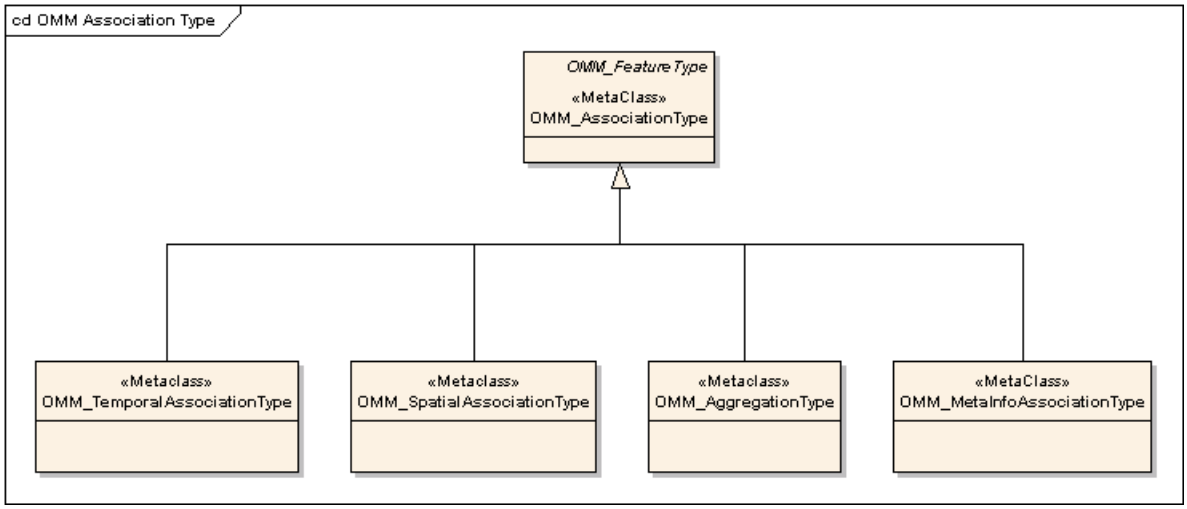

Figure 17: Subclasses of OMM_AssociationType

8.5 Inclusion of the Source System Level

8.5.1 Extension of the Information Model Framework

The RM-OA specifies a service-oriented architecture that is dedicated to the integration of systems providing both information and services (see section 5.4.3). For this purpose, ORCHESTRA offers means and services for syntactic and semantic interoperability. Thus, the RM-OA specifies an architecture for a “system of systems” or “networked systems”. These systems may already exist, whether implemented in older technologies (“legacy systems”) or in more recent technologies, or they may already be built based on ORCHESTRA services.

Regardless of their structure, their technology, their information or their services, these systems are called “source systems” in the sequel. They provide the source of information and services to be integrated into an OSN.

Source systems are of a very heterogeneous nature with respect to their structure and content. Examples of source systems are relational or object-oriented databases, information systems, document archives, map servers, Web sites and sensors. As a consequence, the interfaces to access the information contained in a source system or to call a service offered by a source system are very diverse. Although they are sometimes based on individual de-facto or de-jure standards (e.g. SQL, JDBC/ODBC, CORBA, RMI, Web Services, .NET), there is no standard interface for the integration of source systems as a whole.

Figure 18 illustrates the consequences for the information model framework when explicitly taking the source system level into account.

The majority of source systems do not comply with the ISO, OGC or ORCHESTRA understanding of a feature, nor is their information model specified according to the respective feature models. In order to allow ORCHESTRA services to process this information, data and information of the source systems have to be converted into an OFS according to an OAS. Whether the resulting OFS is persistently stored or just maintained in a transient manner depends on the implementation architecture and the task to be fulfilled. The only requirement on source systems is that (possibly through some software adapter) they may offer their data and/or functions in a way that complies with the OMM.

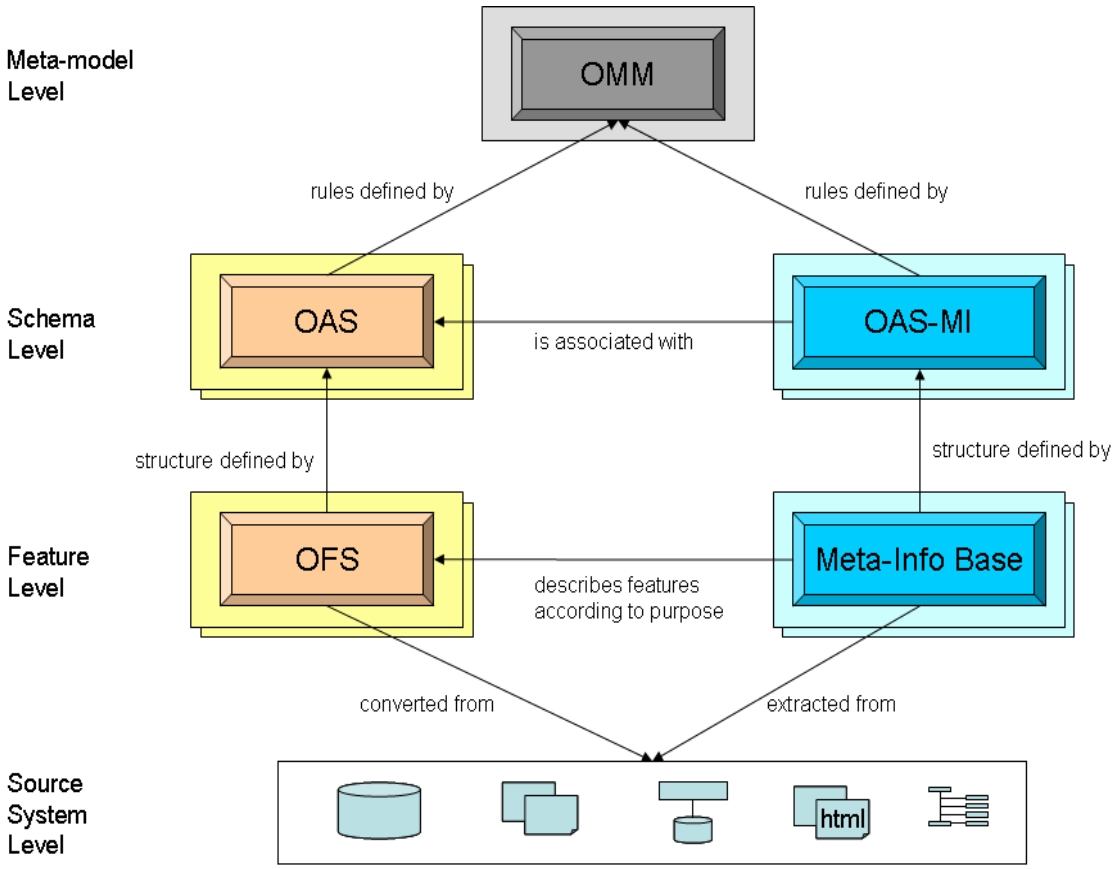

Figure 18: Inclusion of the Source System Level into the ORCHESTRA Information Model Framework

Furthermore, before ORCHESTRA services may access the information of the source systems, they have to be known in an OSN, either by means of an explicit registration step initiated by the source systems or by means of a discovery process initiated by OSN components. For this purpose, meta-information about the source systems, their information and/or their services is required.

This meta-information has to be extracted from the source systems, either by an explicit delivery process initiated by the source systems or their providers, or automatically by some extraction (annotation) process of meta-information initiated by a software component in an OSN. In any case, the extraction of meta-information is guided by the respective OAS-MI specifically designed for this particular purpose.

Note: The process for converting source system information into an OFS and the process for extracting meta-information about source systems for a particular purpose are independent processes. They may be performed in an isolated manner (e.g. just discovery based on provided meta-information), subsequently (e.g. firstly discover the source system using the meta-information provided, and secondly access to the source system information via the OFS) or in parallel (e.g. offline transformation of a source system into an OMM-compliant information system).

8.5.2 Scenario for Data Interchange related to ISO 19109

ISO 19109 specifies two patterns for the interchange of information between systems to be supported:

- Data interchange by transfer: this is the more traditional model where only the data along with the application schema describing its structure are exchanged between the two partners;
- Data interchange by transaction: in this usage pattern, the communication protocol for querying or modifying data is also specified allowing systems to communicate directly.

For the ORCHESTRA Architecture, being a service-oriented architecture, the data-interchange-by-transaction pattern will be used.

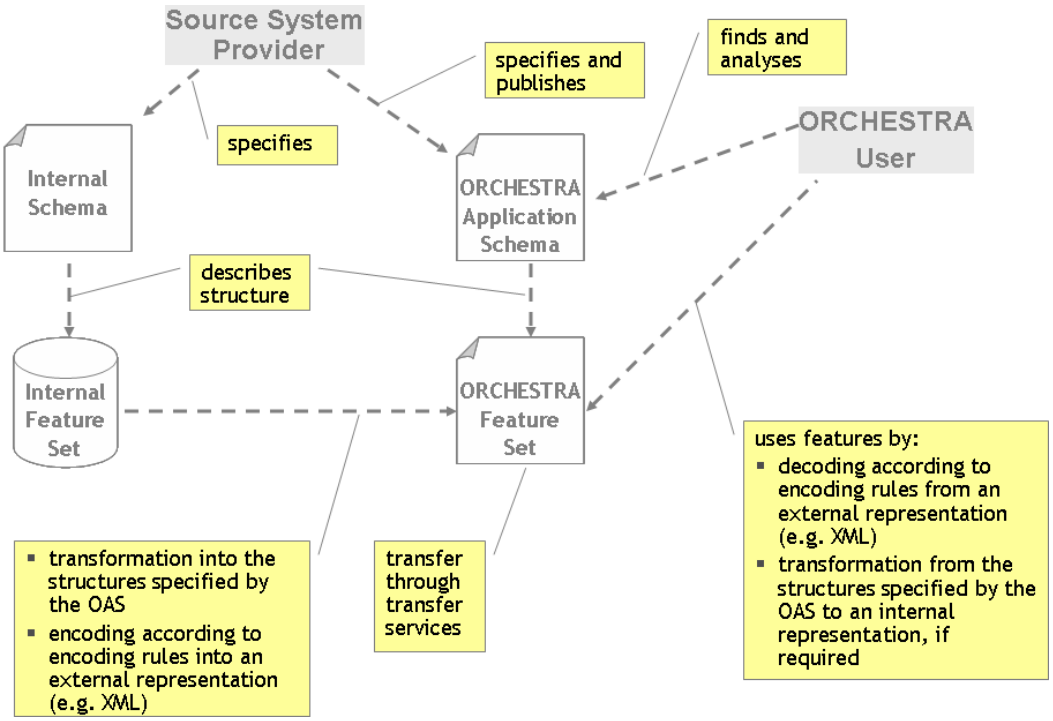

Figure 19: Ad-hoc use of published feature sets and application schemas

The descriptions in ISO 19109 can be read in a way that data interchange according to that International Standard requires agreement of all parties involved in the interchange over the application schema. Within the ORCHESTRA Architecture a typical usage scenario will be that a source system provider will publish its data (OFS) and the application schema describing it (OAS) without consulting most potential users of the data. If a potential user then discovers the OFS/OAS through catalogues, carries out an assessment of the usability of the feature set for his task and decides to use the data, this is then considered as an agreement (ex-post) over the application schema to be used in the data interchange, too.

This scenario is illustrated in Figure 19

8.6 Inclusion of the Semantic Level

8.6.1 Ontologies

The semantic level provides semantics to the information specified in the other levels, e.g. through explicit consideration of ontologies defined and shared in user communities.

An ontology is an explicit, formal specification of a shared conceptualisation (Studer et al 1998). Ontologies may be thought of as a formal representation of the knowledge associated with a particular subject area (domain) or task. Their ultimate purpose is to enable machine understanding, which in turn provides the potential for data and service interoperability.

8.6.1.1 Ontology Classes

Ontologies may be broadly classified as listed in Table 3 (ORCH-D2.3.5 2006). Domain and task ontologies capture knowledge at a level of abstraction free from implementation concerns – that is, they reflect the pure nature of the domain or task. The application and data ontologies are descriptions of information system implementations, and are only necessary if domain and task ontologies cannot be mapped directly to these implementations. Domain ontologies are intended to provide a source of predefined concepts for use with task ontologies. Task ontologies will typically cross domains and therefore draw concepts from more than one domain ontology.

Ontology Class	Definition
Domain Ontology	A formalisation of the knowledge in a subject area (domain) such as topography, ecology, biology, flooding, etc.
Task Ontology	A formalisation of the knowledge necessary to solve a specific problem or task but abstracted above the level of a specific situation or organisational context, for example performing the task of monitoring fresh water quality.
Application Ontology	Contains knowledge for a specific application designed to complete a task in a specific situation and organisational setting, such as the task of monitoring water quality as performed by the Environment Agency. Such ontologies will contain little knowledge that is directly reusable by other organisations and serve to provide a semantic interface between the domain and task ontologies and the application.
Data or Service Ontology	Describes a service or data source and may be seen as a special type of an application ontology.

Table 3: Ontology Classes (ORCH-D2.3.5 2006)

Within the RM-OA, ontologies of these classes may be taken into account as follows:

- Domain Ontologies may be used in order to provide a semantic reference for ORCHESTRA Information Models and ORCHESTRA Meta-Information Models.
- Task Ontologies may be used in the context of service chaining and workflow modelling and will be considered as part of the RM-OA Service Viewpoint specification.
- Application and Data Ontologies may be used to support the integration of source systems. Here, available application or data ontologies are meta-information for the source systems. Thus, they will be considered as part of the RM-OA Information Viewpoint in the context of
  - the schema mapping between internal schemas of source systems and respective OAS, or
  - the process of converting data from source systems into OFS according to an OAS, or
  - the process of extracting meta-information from source systems.
- Service Ontologies may also be used to support the integration of source systems with a particular focus on the discovery and mediated access to services provided by source systems. Here, service ontologies are meta-information for the services of source systems. Thus, they will be considered as part of the RM-OA Information Viewpoint in the context of the process of extracting meta-information from source systems. Their usage for the service mediation will be specified as part of the RM-OA Service Viewpoint.

Note 1: The RM-OA will start with the consideration of domain ontologies. Domain ontologies are the most advanced ones in the research community of the Semantic Web. Furthermore, they play a major role within the ORCHESTRA project (ORCH-D2.3.5 2006).

Note 2: The current version of the RM-OA has its focus on the support of syntactic interoperability. Thus, this RM-OA version just positions domain ontologies in the framework for ORCHESTRA Information Models. Version 3 of the RM-OA will provide more detailed specifications of how ontologies influence the RM-OA Information and Service Viewpoints.

8.6.1.2 Conceptual and Logical Ontologies

Ontologies are formal representations of the knowledge associated with a particular subject area (domain) or task, whose ultimate purpose is to enable machine understanding of the knowledge in a particular domain (ORCH-D2.3.5 2006). Within the RM-OA, ontologies are considered in two appearances according to the following two development stages of ontologies:

- The first stage is the construction of a conceptual ontology by the domain expert. A conceptual ontology is structured knowledge in a domain which a domain expert can understand. Its

documentation includes the following:

- A glossary of concepts, instances, relationships, their natural language definitions, assigned characteristics and values, and additional information assigned to the relationships.
- Sources of the documents used to create the content of the glossary.
- Defined rules, assumptions and primitives used to express the definitions.
- Concept networks and hierarchies (either in a diagrammatic format or in linear notation).
- Relationship networks and hierarchies (either in a diagrammatic format or in linear notation).
- Defined rules and assumptions regarding the networks or hierarchies.

- The second stage is the transformation of the structured knowledge base into a machine-readable logical ontology by an ontology expert. The resulting logical ontology is thus defined in a machine-readable notation like e.g. OWL.

8.6.1.3 High-level Ontologies

A high-level ontology could be expected to contain terms of a more abstract nature or coarser level of granularity that can be related (through subsumption relationships) to those concepts in other domain ontologies which capture knowledge at a finer level of granularity (ORCH-D2.3.5 2006). For example in the thematic context of risk management, a “flood risk” domain ontology may include concepts like “flood risk map”, “risk of flood”, and “velocity measurements”, and may need to use their super-ordinate, more generic terms, to effectively describe these concepts. The super-ordinate generic concepts are, however, often out of scope. A high-level ontology serves the purpose of containing these generic terms which are common across several domains. A high-level ontology, which the “flood risk” ontology could reuse, would contain concepts such as “map”, “risk”, and “river data”.

Due to the generic nature of the RM-OA, those generic concepts of high-level ontologies that are not tied to a particular thematic domain have the highest relevance to be considered as basic information elements in the framework of ORCHESTRA information models (see section 8.4).

8.6.2 Extension of the Information Model Framework for Domain Ontologies

The extension of the information model framework after domain ontologies have been taken into account is illustrated in Figure 20.

As mentioned above, the RM-OA distinguishes between conceptual and logical ontologies. This is reflected in the framework on the semantic level whereby the logical ontology is the result of a transformation process from the conceptual ontology.

As the RM-OA specifies a generic ORCHESTRA Architecture, the information viewpoint is not tied to a specific domain ontology either on the conceptual or on the logical level.

Note: The handling of the conceptual model and the transformation process to the logical ontology is out of scope of the RM-OA. The RM-OA Version 3 will discuss the aspects of semantic interoperability based on machine-processable logical ontologies.

Examples of relationships to the other levels of the specification framework are illustrated in Figure 20:

ex 1. Generic concepts that are relevant across a multitude of domain ontologies (possibly collected in form of a high-level ontology) are candidates for the specification of additional meta-classes in the OMM. Examples here are documents or maps.

ex 2. An OAS-MI provides an application schema for meta-information for a particular purpose. Usually, the classes and their characteristics in the form of attributes and operations used in the application schema have no formally defined semantics. In order to support mediation tasks using the meta-information, the concepts in a domain ontology including their natural language definition (i.e. the glossary) could be referred to by the classes in the OAS-MI.

ex 3. OAS may be generated from logical ontologies if these have a sufficient level of detail, e.g. if they include typed slot definitions that may be mapped to feature properties types.

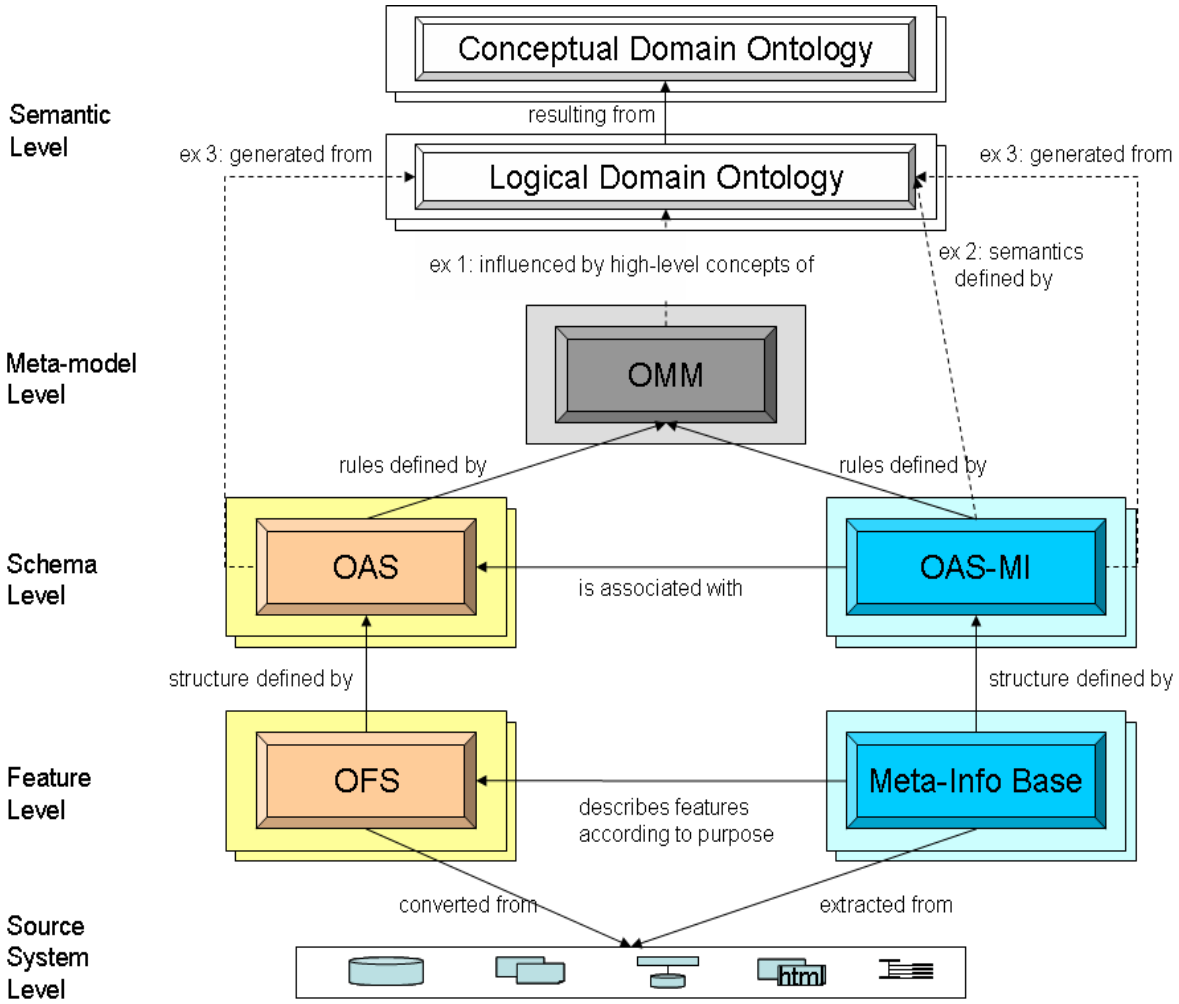

Figure 20: Inclusion of the Semantic Level into the Information Model Framework

8.7 The ORCHESTRA Meta-Model for Information

8.7.1 Overview

As mentioned above, the OMM is derived from the basic ideas of the ISO 19109 GFM, but it is not a true profile of it. In particular, the GFM requires that

- all data quality attribute types are implemented using DQ_Element as specified by ISO 19115,
- all “GFM metadata” attribute types are implemented using “metadata classes” as specified by ISO 19115, and
- a “GFM metadata element” has to be used as a GF_Metadata_AttributeType to carry “metadata” about instances of feature types.

Note: The term “metadata” here refers to its meaning and usage in ISO 19109 and ISO 19115.

While this may be true in a particular OAS, an OAS is *not* required to adhere to these rules. For instance, ORCHESTRA application schemas for meta-information will have to support other standards and other information models. See section 8.4 for additional details.

This is why the OMM is an evolution of the ISO 19109 GFM, taking into account additional, ORCHESTRA-specific requirements. After defining the data types to be used in the OMM and ORCHESTRA application schemas in section 8.7.2, the OMM is specified in two steps:

- the OMM selects the classes and properties of the GFM that are relevant for ORCHESTRA (see sections 8.7.3 and 8.7.4)
- the OMM adds additional meta-classes, namely for additional meta feature and attribute types (see sections 8.7.4 and 8.7.5). Note that the creation of these meta-classes is not strictly required, but shall clearly highlight and list the important information types required by ORCHESTRA applications.

8.7.2 Data Types

8.7.2.1 Introduction

The following section defines the most fundamental data types available in the ORCHESTRA framework. In order to achieve interoperability a common basis is made available and well-defined. ORCHESTRA Basic Data Types (and OA_Types) are part of such a basis.

All data types used and defined in ORCHESTRA shall be built directly and/or indirectly (e.g. OA_Types) using Basic Data Types. This enables ORCHESTRA users to have only one definition for a single type instead of a multitude of definitions (e.g. every service developer and/or every application designer defining its own types for equal purposes). ORCHESTRA basic data types relate and refer to definitions in already accepted standards (like ISO 191xx series) and therefore they are well-known in the software development community.

8.7.2.2 Basic Data Types

Basic Data Types have a standardised definition outside of ORCHESTRA documents (e.g. ISO 191xx series). The names of these types will not be prefixed and refer to standard types. They are defined in Table 4 with the related standard document being referred to in the *Origin* column.

Note: Basic Data Types must not be confused with the UML stereotype called <<DataType>> (see section 8.8.6).

8.7.2.3 OA_Types

OA_Types are predefined types in the OMM which do not have a standardised definition outside of ORCHESTRA documents. They are composed of ORCHESTRA Basic Data Types and other already defined OA_Types. OA_Types might still be rather simple.

8.7.2.4 User-defined types

User-defined types are not predefined within the OMM. They usually refer to types defined for a specific application (e.g. in an OAS) and may only consist of well-known types. These well-known types are Basic Data Types, OA_Types and already specified User-defined types.

Type Names	Origin	Brief Description
Real	ISO/TS 19103 section 6.5.2.5	A signed real (floating point) number consisting of a mantissa and an exponent. (not necessarily the exact value as the common implementation of a Real type uses base 2)
Integer	ISO/TS 19103 section 6.5.2.3	A signed integer number. Exact with no fractional part.
Decimal	ISO/TS 19103 section 6.5.2.4	A number type that represents an exact value as a finite representation of a decimal number. (Unlike real, it can represent 1/10 without error)
Binary	ISO19118 section A.5.2.1.14	Finite-sequence of arbitrary binary data.
Any	ISO/TS 19103	The root of all classes. Often not an actual class in the implementation, it essentially is used where the target class of a member name is not known.
CharacterString	ISO/TS 19103 section 6.5.2.7	Type representing a simple string. The whole string has a single specific encoding. This encoding is retrievable from the string.
CountryCode	As specified by ISO/TS 19139	List of country identifiers.
LanguageCode	As specified by ISO/TS 19139	List of language identifiers.
CharacterSetCode	ISO/TS 19103 section 6.5.2.7	List of character encodings.
MD_Character SetCode	As defined in ISO 19115	List of character encodings.
PT_Locale	As specified by ISO/TS 19139	Type combining language, country and encoding.
Localised CharacterString	As specified by ISO/TS 19139	A CharacterString with the addition of a field specifying the language of the string.
Enumeration	ISO/TS 19103 section 6.5.4.2	Defined and closed list of valid mnemonic identifiers.
CodeList	ISO/TS 19103 section 6.5.4.3	An open Enumeration.
Boolean	ISO/TS 19103 section 6.5.2.11	A value specifying TRUE or FALSE
Date	ISO/TS 19103 section 6.5.2.8	Type representing a date.
Time	ISO/TS 19103 section 6.5.2.9	Type representing a point in time.
DateTime	ISO/TS 19103 section 6.5.2.10	Type combining date and time.
Set	ISO/TS 19103 section 6.5.3.2	Unordered finite collection of non duplicate objects.
Bag	ISO/TS 19103 section 6.5.3.3	Unordered finite collection of possibly duplicate objects.
Sequence	ISO/TS 19103 section 6.5.3.4	Ordered ‘bag-like’ structure.
Dictionary	ISO/TS 19103 section 6.5.3.5	Container for key-value pairs where the key and value types are not predefined.

Table 4: Basic Data Types

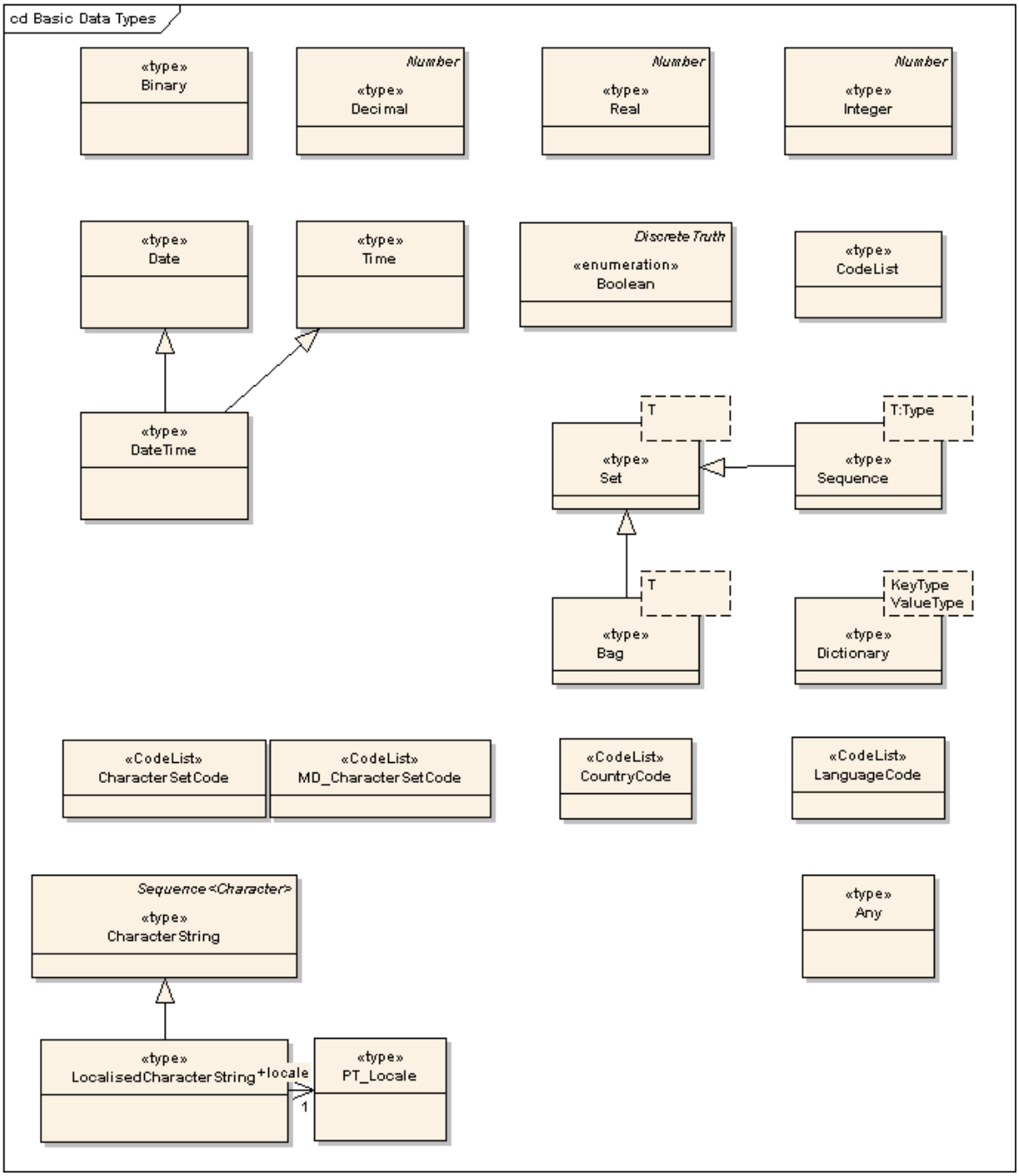

Figure 21: Basic Data Types

8.7.3 OMM Basic Part

The UML class diagrams in Figure 22 show the basic part of the OMM that principally specifies the relationship between OMM_FeatureTypes, OMM_PropertyTypes and OMM_AssociationTypes. It exactly corresponds to the main structure of the GFM as described in the section 7.3.3 (GFM main structure), section 7.3.4 (GF_FeatureType) and section 7.3.5 (GF_PropertyType) and illustrated in Figure 5 of the ISO 19109 GFM document.

The meaning of the respective meta-classes prefixed by OMM_ is the same as the meaning of the meta-classes prefixed by GF_ in ISO 19109 GFM.

The extension of the OMM with respect to the GFM relates to the extended understanding of what a feature type could be in ORCHESTRA as described section 8.2.

Note: The architectural principle of "self-describing components" (see section 6.3.7) requires that there is a means to get the feature type specification for a given feature instance. This principle is indirectly fulfilled through the capabilities (see section 9.2.5.2) of the service instance that makes the feature instance available.

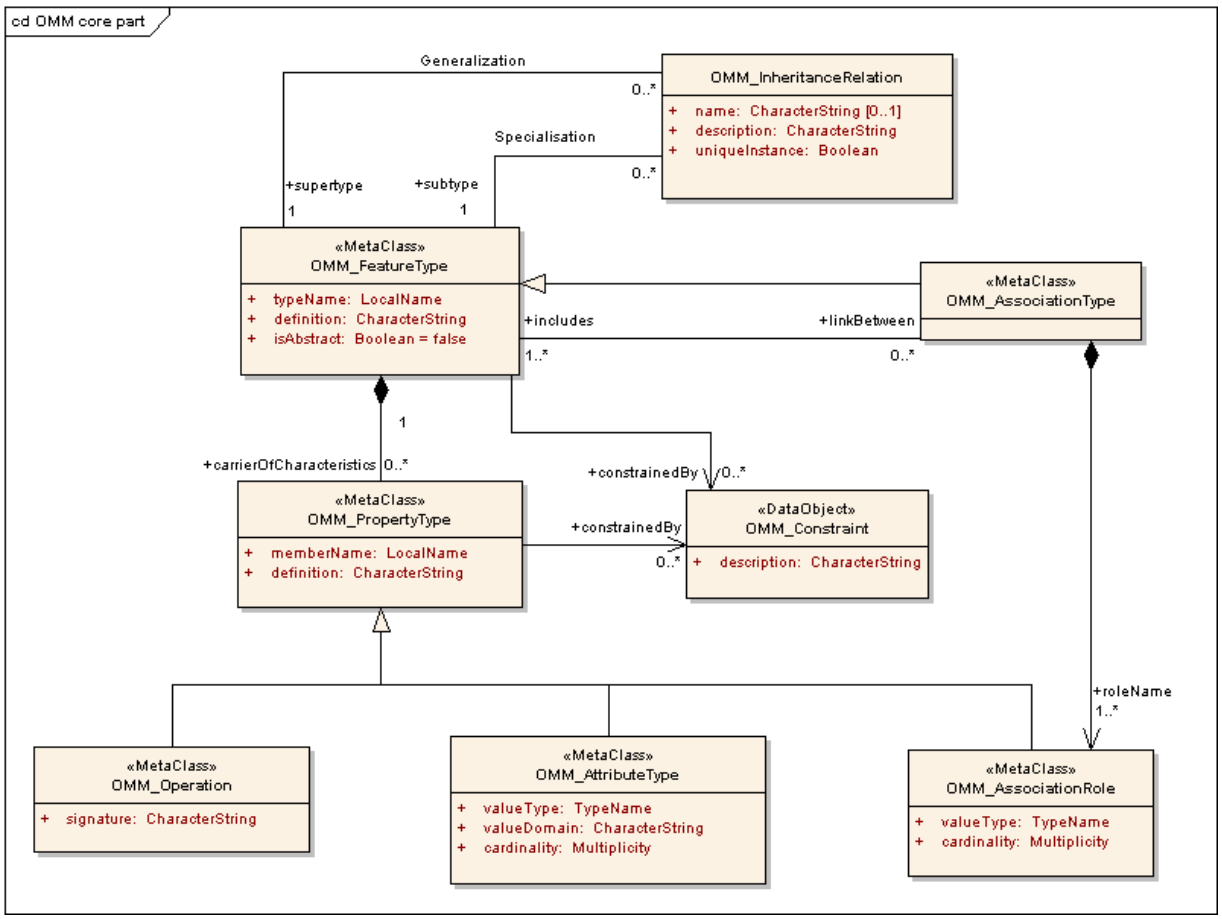

Figure 22: The basic part of the ORCHESTA Meta-model

8.7.4 OMM Attribute Types

The ORCHESTRA Architecture uses the following categories of attribute types and their base class from the ISO 19100 series:

- Spatial Geometry (ISO19107::GM_Object)
- Spatial Topology (ISO19107::TP_Object)
- Temporal Object (ISO19108::TM_Object)
- Geographic Identifier (ISO19112::SI_LocationInstance)
- Data Quality Information (ISO19115::DQ_Element) (see note 1 below)
- Metadata (ISO19115::MD_Metadata) (see note 2 below)

Note 1: The modelling of data quality information or meta-information in the form of attribute types as further specified in ISO 19115 is just one possibility for a meta-information model and the specification of meta-information in the context of an OAS. ORCHESTRA does support further types of meta-information models depending on the particular purpose of the usage of the meta-information (see section 8.4.1).

Note 2: The OMM does not specify meta-information attributes as a prominent high-level attribute type category. Instead, the modelling of meta-information attribute types (OMM_MetaInfoAttributeTypes) as a meta-class that specialises the meta-class

OMM_ThematicAttributeType means that a thematic attribute may use type definitions of ISO 19115 as data type values. See also Rule 1 in section 8.8.11

The resulting schema is illustrated in UML in Figure 23.

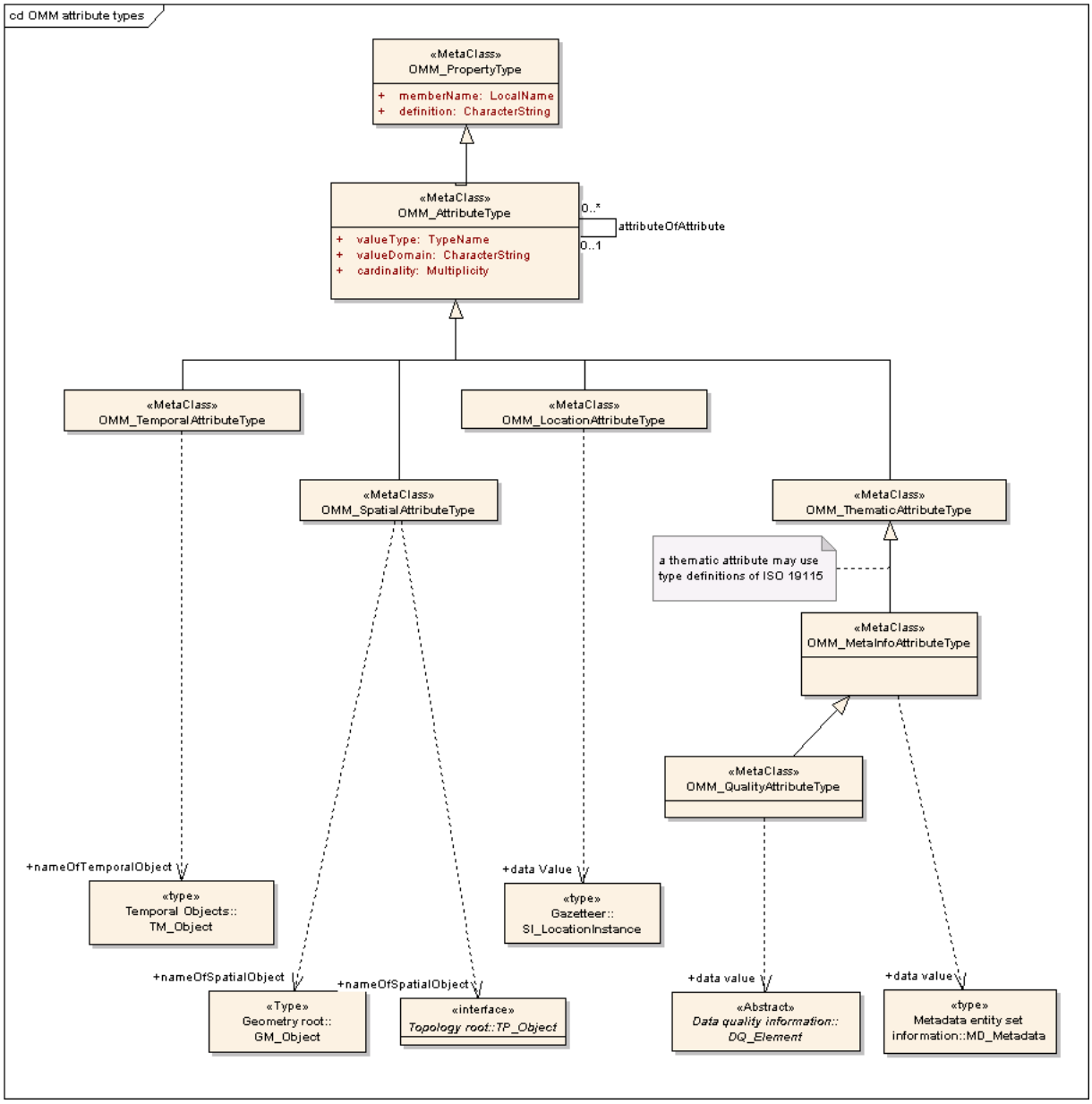

Figure 23: OMM Attribute types

8.7.5 OMM Extensions to Feature Types

8.7.5.1 Overview

As will be defined in the rules below (see section 8.8), an ORCHESTRA Feature Type is defined by a UML class that is part of an OAS as an instance of the OMM meta-class “feature type”. Within an OAS, it has a stereotype “FeatureType”.

Feature types are defined by an information modeller or, in some specific cases, on-the-fly by a software component of an ORCHESTRA Application as part of an OAS and represent “abstractions of real-world phenomena perceived in the context of an ORCHESTRA Application”.

Based on the requirements of thematic domains, the OMM extends the OMM_FeatureType definition

for additional categories of information types. As a result of an analysis of the requirements of the risk management thematic domain that took place in the ORCHESTRA project, the following eminent but generic information types have been identified:

- Document Descriptor type (see section 8.7.5.2)
- Coverage type (see section 0)

Note: By intention, this list of predefined feature types is kept very restrictive in order not to move too much domain-specific information into the meta-model and thus to endanger reusability in different application domains.

8.7.5.2 Document Descriptor Type

Documents are resources that contain recorded information and can be treated as unit. As pre-defined ORCHESTRA feature type, a document is represented by a document descriptor that contains identification information (such as name and document type) and a reference to one of more files (the document store) if the document data is stored locally or a reference to a source system if the document data is stored remotely.

An instance of OA_ThematicAttributeType may represent an attribute that carries document information. The value-types of document attributes shall comply with the definition of an OA_DocumentDescriptor as defined below.

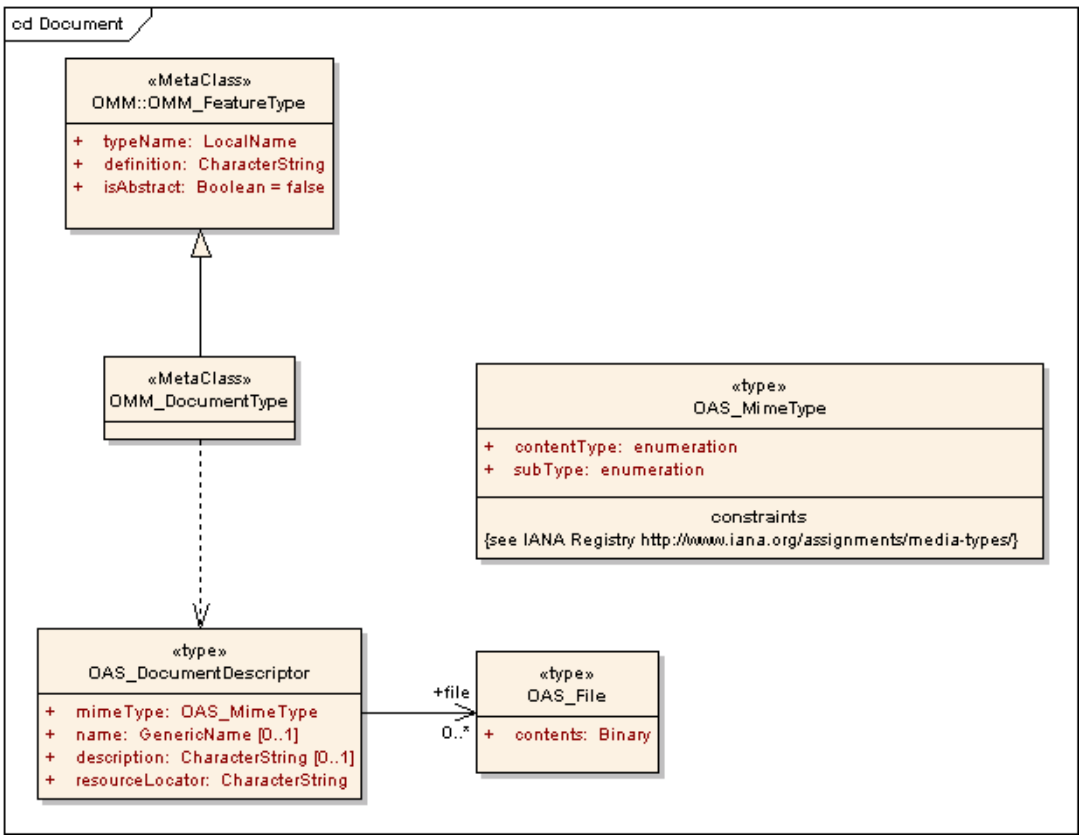

Figure 24: Schema of the OMM extension "Document Type"

Document types may be classified according to the MIME Media Types and include e.g.

- Documents with page layout (e.g. PDF, MS-Word, MS-PowerPoint files, Web pages based on html)
- Audio files
- Video files

- Image files
- XML documents
- tabular data in file format (e.g. an MS-Excel file)

The document schema used in ORCHESTRA is specified in Figure 24.

8.7.5.3 Coverage Type

A coverage denotes a function from a spatial, temporal or spatiotemporal domain to an attribute range. A coverage associates a position within its domain to a record of values of defined data types. Thus, a coverage is a feature with multiple values for each attribute type, where each direct position within the geometric representation of the feature has a single value for each attribute type. Examples include a raster image, polygon overlay, or digital elevation matrix.

The coverage model is defined by ISO 19123.

The domain of a coverage is a set of geometric objects described in terms of direct positions, which are associated with a spatial or temporal coordinate reference system. Commonly used domains include point sets, grids, collections of closed rectangles, and other collections of geometric objects. The range of a coverage is a finite or a transfinite set of feature attribute values.

Coverages can be discrete or continuous. A discrete coverage has a domain that consists of a finite collection of geometric objects and the direct positions contained in those geometric objects. A discrete coverage maps each geometric object to a single record of feature attribute values. A continuous coverage has a domain that consists of a set of direct positions in a coordinate space, which it maps to value records. It then returns a distinct record of feature attribute values for any direct position within its domain.

Note: The term coverage may be misleading as it implicitly refers to a 2-dimensional data layer. The term field would be better as it refers to n-dimensional data. However, the term coverage is used in order to conform with ISO 19123.

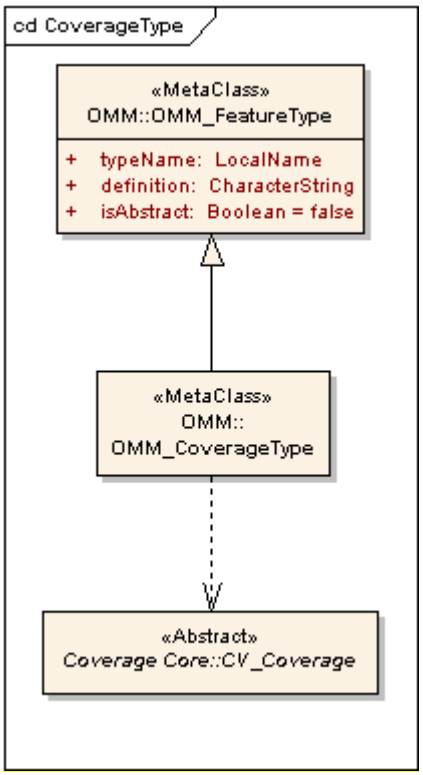

Figure 25: Schema of the OMM Extension "Coverage Type"

8.8 Rules for ORCHESTRA Application Schemas

8.8.1 General Approach

The modelling process for OAS on the platform-neutral level corresponds to the description in ISO 19109, section 8.1. This approach allows automatic derivation of platform-specific application schemas (e.g. GML Application Schemas according to ISO 19136) from the conceptual application schemas in a normative way. GML Application Schemas can be used to encode ORCHESTRA feature instances in XML. GML is tightly integrated with most OGC Web Service specifications, e.g. the Web Feature Service. In addition, mapping to other platforms is possible from the conceptual UML model.

Note: The relationship to the rules for application schemas as specified in ISO 19109, section 8, (conformance, changes and/or extensions) is explicitly indicated in respective notes.

Rules:

1) The data structures of the application shall be modelled in the OAS.

   Note: Rule conforming to ISO 19109, section 8.2.2, rule 1).

2) An abstract specification of an OAS shall use UML 2.0 as its conceptual schema language following the rules of ISO/TS 19103 and ISO 19109. It shall be documented using class diagrams.

   Note: ISO/TS 19103. Geographic information - Conceptual schema language is still based on UML 1.3. A potential conflict will have to be resolved in dedicated rules.

3) An OAS shall use the UML extensibility mechanisms “stereotypes” and “tagged values” as described in annex D.8 of ISO/TS 19103.

   Note 1: A stereotype is a model element that is used to classify (or mark) other UML elements so that they in some respect behave as if they were instances of new virtual or pseudo meta-model classes whose form is based on existing base meta-model classes. Stereotypes augment the classification mechanisms on the basis of the built-in UML meta-model class hierarchy. Therefore, names of new stereotypes must not clash with predefined meta-model elements or other stereotypes. See section 8.8.6 for the rules how to use stereotypes in an OAS.

   Note 2: A tagged value is a tag-value pair that can be used to add properties to any model element in UML, i.e. it can extend an arbitrary existing element in the UML meta-model or extend a stereotype.

8.8.2 Rules for the Identification of an OAS

Rules:

1) The identification of each application schema shall include a name and a version. The inclusion of a version ensures that a supplier and a user agree on which version of the application schema describes the contents of a particular dataset.

   Note 1: This rule conforms to ISO 19109, section 8.2.3, rule 2).

   Note 2: The agreement between supplier and user also covers the case where there is no explicit bilateral agreement, but where the user is able to discover and understand which version(s) of an application schema are supported by the supplier.

   Note 3: It is recommended that the name of an OAS be globally unique (e.g. an URI) in order to enable unambiguous re-use of its elements in other OAS.

2) In UML, an application schema shall be described within a PACKAGE, which shall be stereotyped with <<Application Schema>> and shall contain the tagged value “OAS” carrying the name of the application schema and the tagged value “version” carrying the version stated in the documentation of the PACKAGE.

Note 1: This rule extends ISO 19109, section 8.2.3, rule 1).

Note 2: An OAS may consist of several hierarchically ordered packages. In this case, the OAS name corresponds to the name of the top-level package.

8.8.3 Rules for the Documentation of an OAS

Rules:

1) An OAS shall be documented.

   Note: This rule conforms to ISO 19109, section 8.2.4, rule 1).

2) The documentation of an OAS shall include a reference to the version of the RM-OA that has been used by setting the tagged value “RM-OA” to the version number of the RM-OA document.

3) The documentation of an OAS in UML may utilise the documentation facilities of the software tool that is used to create the application schema, if this information can be exported.

   Note: This rule conforms to ISO 19109, section 8.2.4, rule 2).

4) Documentation of the elements in the UML model shall be stored in tagged values “documentation”.

5) If a CLASS or other UML component corresponds to information in a feature catalogue, the reference to the catalogue shall be documented.

   Note: This rule conforms to ISO 19109, section 8.2.4, rule 3).

6) Documentation of feature types in an OAS shall be in a catalogue with a structure derived from OMM, for instance in a catalogue in accordance with ISO 19110

   Note: This rule conforms to ISO 19109, section 8.2.4, rule 4).

8.8.4 Rule for the Integration of an OAS and other Schemas

Rules:

1) An OAS can be built up of several other application schemas. Each of these schemas can refer to standardised schemas. This organisation can be used to avoid the creation of large and complex schemas (see ISO 19109, section 8.2.6).
2) The dependency mechanism in UML shall be used to describe the integration of the OAS with other application schemas or other standard schemas that are required to form the complete definition of the data structure.

   Note: This rule is derived from ISO 19109, section 8.2.5, rule 1).

8.8.5 Rules for the Usage of Types in an OAS

Rules:

1) Basic Data Types as specified in section 8.7.2.2 and OA_Types as specified in section 8.7.2.3 shall be used where applicable.
2) Types defined in OA Services (see section 9.3.2) shall be prefixed by OA_.

   Note: An example is the OA_GetCapabilitiesRequest type defined in the *ServiceCapabilities* interface type (see section 9.6.1).
3) Types defined in OT Services (see section 9.3.3) shall be prefixed by OT_.
4) An OAS designer is not enjoined to use prefixes for the specification of user-defined types (e.g.

in an OAS), however, OA_ and OT_ are excluded.

8.8.6 Rules for the Usage of Stereotypes in an OAS

Rules:

1) Every class in an application schema must be stereotyped. The stereotype used must be defined either in the standard UML or the stereotypes defined within the OMM. If the stereotype has a name common to the names of those stereotypes already specified, the definition (meaning) has to be the same.

   Note: This facilitates the understanding of OAS and supports application development, e.g., to help decide whether a class is a feature type or not.

2) Data types shall be modelled as UML classes with the stereotype <<DataType>>.

   Note: According to ISO/TS 19103 a <<DataType>> is a descriptor of a set of values that lack identity (independent existence and the possibility of side effects). The primary purpose of a DataType is thus to hold the abstract state of another class (e.g. a class representing a feature type) for transmittal, storage, encoding or persistent storage. An example in the OMM is the aggregation of operation request parameters in one class (see section 9.2.8).

3) Types shall be modelled as UML classes with the stereotype <<Type>>.

   Note 1: According to ISO/TS 19103, a <<Type>> is a stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A type may have attributes and associations.

   Note 2: For the definition of the types and their classification see section 8.8.5.

4) Enumerations shall be modelled as UML classes with the stereotype <<Enumeration>>.

   Note: See section 8.8.5 for the definition of an enumeration as a basic type in an OAS.

5) Code lists shall be modelled as UML classes with stereotype <<CodeList>>.

   Note 1: According to ISO/TS 19103, a code list can be used to describe an open enumeration (see rule 4 above). This means that it needs to be represented in such a way that it can be extended during system runtime.

   Note 2: See section 8.8.5 for the definition of an enumeration as basic type in an OAS.

6) Interfaces shall be modelled as UML classes with stereotype<<Interface>>.

   Note: See the corresponding rule of the OMM-Service in section 9.2.6.

8.8.7 Rules for the Specification of an OAS

Rules:

1) All classes used within an OAS for data transfer shall be instantiable. This implies that the integrated class must not be stereotyped <<interface>>.

   Note: This rule conforms to ISO 19109, section 8.2.2, rule 2).

2) All package names used within an OAS shall be unique.

3) Dependencies between packages must be modelled explicitly.

4) If a class is a specialization of another class, then this class shall have one of the stereotypes <<FeatureType>>, <<DataType>>, or <<Type>>. The class shall have zero or one supertype with the same stereotype and zero or more abstract supertypes of the stereotype <<Type>>.

   That is, disregarding abstract classes with stereotype <<Type>>, a generalization relationship may be specified only between two classes that are either:
   - both feature types (stereotype <<FeatureType>>),
   - both types with stereotype <<Type>>, or
   - both data types (stereotype <<DataType>>).

   For every abstract class <<Type>> all direct or indirect subtypes must be either
   - all classes with stereotypes <<FeatureType>>, <<Type>>, or
   - all classes with stereotypes <<DataType>> or <<Type>>, where all <<Type>> classes have to be abstract.

   All generalization relationships between classes shall have no stereotype. All generalization relationships with other stereotypes will be ignored. The discriminator property of the UML generalization shall be blank.

5) OMM_FeatureType: An instance of OMM_FeatureType shall be implemented as a CLASS stereotyped with <<FeatureType>> except for Rule 6 (see OMM_AssociationType below).

   Note: This rule extends ISO 19109, section 8.3.1, rule 1).

6) OMM_AssociationType: An instance of OMM_AssociationType shall not be associated with any instances of OMM_PropertyType. It has the role of linkBetween in associations to those instances of OMM_FeatureType that are being implemented as CLASSes.

   Note 1: This rule conforms to but restricts ISO 19109, section 8.3.1, rule 2).

   Note 2: This rule means that attributed associations between feature types (i.e. associations with own properties) are not supported.

7) OMM_AggregationType: An instance of OMM_AggregationType shall either be implemented as an AGGREGATION (empty diamond) or it shall be implemented as a COMPOSITION (filled diamond). Members of an aggregation can exist independently of the aggregate, and may belong to other aggregates. Members of a composite may not exist independently and may belong to only one composite.

   Note: This rule conforms to ISO 19109, section 8.3.1, rule 3).

8) OMM_AttributeType: An instance of OMM_AttributeType shall be implemented as an ATTRIBUTE, unless it is an attribute of an attribute (see rule 5)

   Note: This rule conforms to ISO 19109, section 8.3.1, rule 4).

9) attributeOfAttribute: An instance of OMM_AttributeType that acts in the role characterizedBy in an attributeOfAttribute association shall be instantiated as a class with a valid stereotype for classes (e.g., <<FeatureType>>). That class shall be used either as the data type of the OMM_AttributeType, or in an association with the class that contains the OMM_AttributeType. Attributes that act in the role characterizes shall be instantiated as attributes of the class that represents the attribute that acts in the role characterizedBy.

   Note 1: This rule extends ISO 19109, section 8.3.1, rule 5).

   Note 2: This means that a class stereotyped as <<FeatureType>> may be used as a data type of an attribute in a class definition

10) OMM_Operation: An instance of OMM_Operation shall be implemented as an OPERATION of the class representing the feature type that it characterizes, which shall have ASSOCIATIONS to other CLASSES from which the operation needs ATTRIBUTE VALUES.

    Note 1: This rule conforms to ISO 19109, section 8.3.1, rule 6).

    Note 2: The relationship between an operation specified in a feature type and operations specified in interface types (i.e. the link to the OMM-Service meta-classes) will be investigated in a future version of the RM-OA.

11) OMM_AssociationRole: An instance of OMM_AssociationRole shall be implemented as a role name at the appropriate end of the ASSOCIATION representing the OMM_AssociationType.

    Note: Rule conforming to ISO 19109, section 8.3.1, rule 7).

12) OMM_InheritanceRelation: An instance of OMM_InheritanceRelation shall be represented by a UML GENERALIZATION relationship, with the following additional characteristics: If uniqueInstance is .TRUE., the {disjoint} constraint shall be attached to the generalization relationship.

    Note: This rule is derived from ISO 19109, section 8.3.1, rule 8).

13) OMM_Constraint: Constraints may be stated in OCL or in plain language and attached to the CLASS, OPERATION or RELATIONSHIP that is constrained. A formal specification of constraints is required when automatic processing is intended.

    Note: This rule extends ISO 19109, section 8.3.1, rule 9).

8.8.8 Rules for Adding Information to a Standard Schema

Rule:

1) If it is necessary to extend or restrict a CLASS specified in a standard schema, a new CLASS shall be defined as a SUBTYPE of the CLASS in the standard schema, and ATTRIBUTEs shall be added to this CLASS to carry the additional information.

   Note 1: This rule conforms to ISO 19109, section 8.4.2, rule 1).

   Note 2: For practical reasons the new classes may be collected in a separate PACKAGE.

8.8.9 Rules for restricted Use of Standard Schemas

Rules:

1) Specification of a restricted profile of a standard schema shall be described in a new UML package by copying the actual definitions (classes and relationships) from the standard schema. Attributes and operations within classes may be omitted.

   Note: This rule conforms to ISO 19109, section 8.4.3, rule 1).

2) Reduction of a standard schema shall be in accordance of the conformance clause given for the actual standard.

   Note 1: This rule conforms to ISO 19109, section 8.4.3, rule 2).

   Note 2: The specifications of OMM extension types (see section 8.7.5) are handled like standard schemas. The rules to be considered for a possible reduction are specified in section 8.8.15.

8.8.10 Rules for Adding Information to an OAS

Rule:

1) If it is necessary to extend a CLASS specified in an OAS, a new CLASS shall be defined as a SUBTYPE of the CLASS in the standard schema, and ATTRIBUTEs shall be added to this

CLASS to carry the additional information.

8.8.11 Rules for Thematic Attributes

Rule:

1) A thematic attribute may reuse definitions from a package in the ISO 19115 without being considered as meta-information in the application schema.

   Note: This rule conforms to the RM-OA approach to handle meta-information (see section 8.4.1). Whether an attribute is to be considered as meta-information cannot be decided at design time.

8.8.12 Rules for Temporal Attributes

Rules:

1) If a common representation of time across systems is required then it is recommended that any description of temporal aspects be in accordance with the specifications given by ISO 19108.

   Note: This recommendation is still to be validated in the course of the ORCHESTRA specification and implementation process, in particular w.r.t. to the usage of the basic data types “date” and “time” as specified in section 8.7.2.2.

2) The usage of temporal attributes according to ISO 19108 in an OAS shall comply with the specifications and rules of ISO 19109, section 8.6, if not otherwise specified in the RM-OA.

   Note: This recommendation is still to be validated in the course of the ORCHESTRA specification and implementation process, in particular in the handling of time-series by the Map and Diagram Service (see section 9.7.2).

8.8.13 Rules for Spatial Attributes

Rules:

1) The value domain of spatial attribute types shall be in accordance with the specifications given by ISO 19107, which provides conceptual schemas for describing the spatial characteristics of features and a set of spatial operators consistent with these schemas. ISO 19125-1 is a profile of 19107 that is widely adopted (see the OGC simple feature specification). If in the process of specifying an OAS there is no explicit need to use other data types than those specified in ISO 19125-1, then ISO 19125-1 shall be used.

   Note: This rule extends ISO 19109, section 8.7, rule 1).

2) The usage of spatial attributes according to ISO 19107 and ISO 19125-1 in an OAS shall comply with the specifications and rules of ISO 19109, section 8.7, if not specified otherwise in the RM-OA.

8.8.14 Rules for Spatial Referencing using Geographic Identifiers

Rules:

1) The value domain of attributes using spatial referencing by geographic identifiers shall be in accordance with the specifications given in ISO 19112.

   Note: This rule conforms to ISO 19109, section 8.9, rule 1).

2) The usage of attributes using spatial referencing by geographic identifiers according to ISO 19112 in an OAS shall comply with the specifications and rules of ISO 19109, section 8.9, if not specified otherwise in the RM-OA.

8.8.15 Rules for Information Types extending the OMM

8.8.15.1 Feature Types vs. Attribute Types

Depending on the semantics, a particular piece of information may be considered either a feature (type) or a value of an attribute (type). When modelling, it is often a judgement call whether to model a particular type one way or the other.

As a general rule, a feature type will be used if the concept is of particular importance for the application, has an identity of its own and can be considered to be an "abstraction of a real world phenomenon."

On the other hand, a concept will be modelled as a data type of an attribute if the concept does not have an identity on its own (i.e. it is just a structured attribute) or if it is just an auxiliary concept and will only be used in the context of a feature (e.g. a geometry or topology object).

8.8.15.2 Rules for Coverages

Coverages are considered in the OMM as instances of ORCHESTRA feature types, see section 8.7.5.2. Their schema is defined in ISO 19123.

Rules:

1) Any description of coverage information shall be in accordance with the specifications given by ISO 19123.
2) A coverage type shall be defined as a coverage feature type which is the appropriate, most specialized type defined in ISO 19123 listed in rule 5 or a subtype of this type.
3) The implementation of a coverage type in UML shall follow the rules (see ISO 19109 8.2.5) for referencing standardised schemas (see RM-OA, section 8.8.4, rule 2).
4) A coverage type shall be represented in an application schema as a UML CLASS that represents a feature (see RM-OA, section 9.2.5.2) and which is derived directly or indirectly from one of the UML classes from rule 5.
5) Valid coverage feature types which shall be applied are::
   - Discrete coverages (CV_DiscreteCoverage)
     - Discrete point coverage (CV_DiscretePointCoverage)
     - Discrete grid point coverage (CV_DiscreteGridPointCoverage)
     - Discrete curve coverage (CV_DiscreteCurveCoverage)
     - Discrete surface coverage (CV_DiscreteSurfaceCoverage)
     - Discrete solid coverage (CV_DiscreteSolidCoverage)
   - Continuous coverages (CV_ContinuousCoverage)
     - Thiessen polygon coverage (CV_ThiessenPolygonCoverage)
     - Hexagonal grid coverage (CV_HexagonalGridCoverage)
     - TIN coverage (CV_TINCoverage)
     - Segmented curve coverage (CV_SegmentedCurveCoverage)
     - Continuous quadrilateral grid coverage (CV_ContinuousQuadrilateralGridCoverage)

   Note: Whether all of these coverage types are required for most of the applications of the RM-OA or if they may be restricted is yet to be determined.

8.8.15.3 Rules for Documents

Documents are considered in the OMM as instances of ORCHESTRA feature types. Their schema is defined in section 8.7.5.2.

Rules:

1) A document type shall be represented in an OAS as an attribute (an instance of OMM_ThematicAttributeType) of a UML CLASS that represents the feature, in which case the attribute shall take OA_DocumentDescriptor as defined in section 8.7.5.2 and Figure 24 or a subtype as the data type for its value.

8.9 A Simple Example

An extremely simplified model of an earthquake feature type is illustrated in Figure 26. In terms of the OMM, the feature type "XE_Earthquake" has the following own properties:

- an optional thematic attribute type with the name "magnitude", the value is a numeric value between 0 and 10 (Richter scale);
- an optional feature association role with the name "officialReport" to a document feature type(see section 8.7.5.2).

Furthermore, by means of multiple inheritance according to the rules specified in section 8.8.7, the XE_Earthquake class inherits the following properties:

- from the feature type “Hazard”: a spatial property type with the name "location", the value type is a spatial point (see ISO 19107).
- from the feature type “Hazard”: a temporal property type with the name "occurredAt", the value type is a temporal instant (see ISO 19108).
- from the type “ObjectWithMetadata”: an optional meta-information property type with the name “metadata”; the value type is a metadata entity (see ISO 19115).

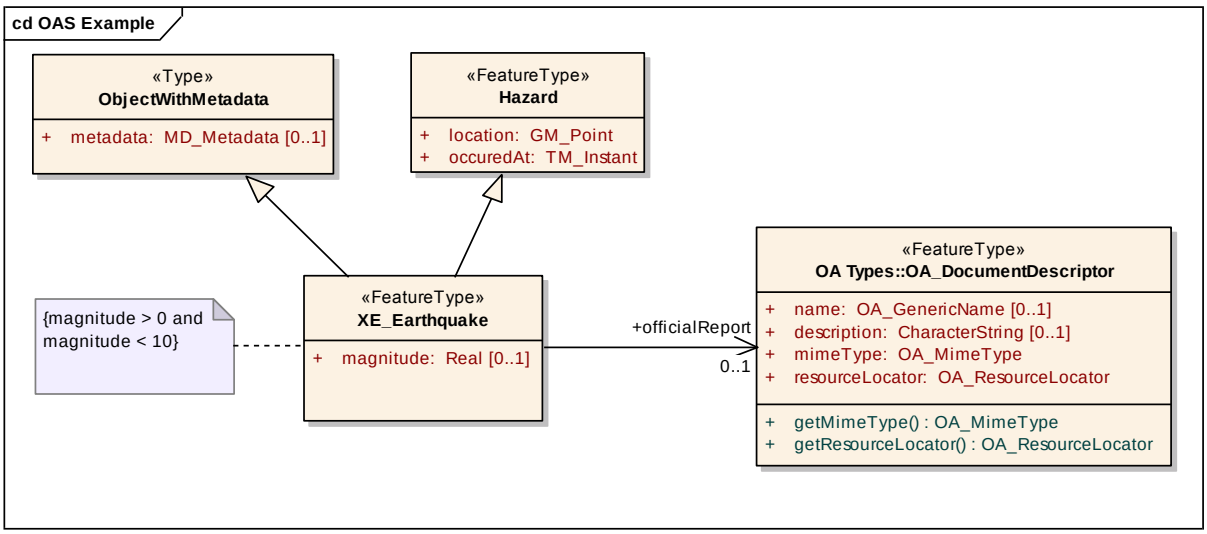

Figure 26: Earthquake example

9 Service Viewpoint

9.1 Overview

The Service Viewpoint of the RM-OA specifies the specification framework for ORCHESTRA Services. This specification framework is provided by the definition of a Service Meta-Model as given in section 9.2.

Furthermore, the Service Viewpoint of the RM-OA provides abstract specifications for the generic ORCHESTRA Services that support the syntactic and semantic interoperability between ORCHESTRA Source Systems and between services and the development of ORCHESTRA Applications. This includes the management of an OSN as one particular application.

In combination with the specification of the ORCHESTRA Information Viewpoint, this specification provides the ORCHESTRA Architecture. According to RM-OA principles, the abstract description of ORCHESTRA Services and the abstract specification of their interfaces include all properties of the services that may be specified in a platform-neutral way. Their mapping to specific service platforms (e.g. a W3C Web Services environment) is outside the scope of the RM-OA and is specified in ORCHESTRA Implementation Specifications.

Section 9.2 provides a Service Meta-model (OMM-Service) as a complementary part of the OMM Information Meta-model (OMM-Information).

ORCHESTRA Services are functionally classified in section 9.3

The RM-OA specifies the ORCHESTRA Services and their interfaces in two different ways:

- A coarse abstract service description is given for each service in human-readable text format by using a service description framework, see section 9.4.
- A refined abstract specification of the interfaces to be realised by the services is given in (ORCH-AbstrServ 2007) by using UML as the conceptual schema language.

Note: Whereas the OMM-Information is an evolution of the General Feature Model (GFM) of ISO 19109 (see section 8.3), the ISO counterpart for the OMM-Service would be the UML model supplied in section 7.2 of ISO 19119 which is, however, not directly related to the GFM. Furthermore, it does not cover the problem of abstract and implementation specification of services. The meta-model approach of ORCHESTRA aims at a harmonised approach for both the information and the service viewpoint with direct interdependencies and rules about how to handle the problem of platform-neutral and platform-specific service specifications and the mapping between them. A need for such an approach has recently been expressed by the Object Management Group (OMG) in their Request For Proposal for a “Software Services Profile and Metamodel” (OMG 2006).

9.2 The ORCHESTRA Meta-Model for Services

9.2.1 Overview

An ORCHESTRA Service is a service specified according to the rules of the ORCHESTRA Reference Model in an ORCHESTRA Service Specification. As with the Information Viewpoint of the RM-OA, these rules are provided by means of a Service Meta-Model as further part of the ORCHESTRA Meta-Model (OMM).

In the Information Viewpoint, the OMM has been defined as the common specification framework for all feature-based application schemas used within ORCHESTRA. It provides a meta-model and a set of associated rules that control the specification of an OAS. This part of the OMM is called OMM-Information in the following. For the Service Viewpoint the schema level is extended by the concept of ORCHESTRA Service Types. The corresponding rules for their specification are defined in a respective extension of the OMM called OMM-Service in the following.

The framework for ORCHESTRA Services is illustrated in Figure 27. It distinguishes between

- the ORCHESTRA Meta-Model (OMM) on the meta-model level,

- ORCHESTRA Service Specifications on the schema level,
- ORCHESTRA Services on the service level and
- the functionality provided by source systems on the source system level.

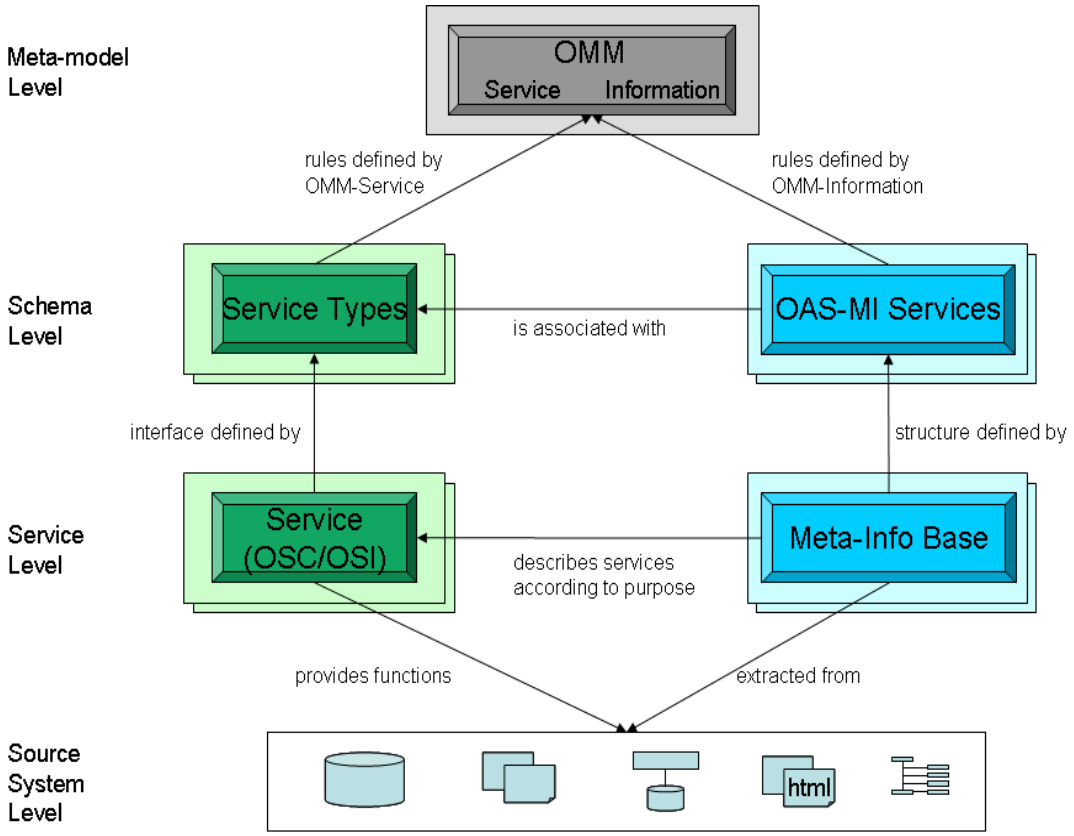

Figure 27: Framework for ORCHESTRA Services

ORCHESTRA Service Types are specified by defining their externally visible behaviour accessible through their service interfaces (see section 9.2.2.3). The service interfaces, including their information models, are expressed using the conceptual schema language UML in the first step (abstract specification), and then mapped to a chosen platform in a second step (implementation specification).

On the schema level, meta-information models are associated to ORCHESTRA Service Types in so-called OAS-MI for Services according to the rules of the Information Viewpoint (OMM-Information) specified in section 8.7. These OAS-MI deliver the schema for the meta-information that is associated with service types in order to serve the various purposes (e.g. discovery of services) as outlined in section 8.4.2.

The service level is built by the set of ORCHESTRA Services and the meta-information base as the logical aggregation of the meta-information that describes the ORCHESTRA Services according to the various purposes. The meta-information base is structured according to the OAS-MI specified on the schema level. ORCHESTRA Services are instances of ORCHESTRA Service Types and have two different appearances:

- as ORCHESTRA Service Components (OSC) when referring to the software component that implements the interfaces defined for the ORCHESTRA Service Types on the schema level, and
- as ORCHESTRA Service Instances (OSI) when referring to deployed and running instances of OSCs in an OSN.

In the Service Viewpoint, the source system level consists of the set of source systems whose functionality is to be integrated into an OSN. For this purpose, source system-specific service types have to be specified by the system integrator and instantiated as OSIs such that the functions of the source sys-

tems may be offered to ORCHESTRA Applications in an ORCHESTRA-compliant way. Note that there is no generic ORCHESTRA Service Type defined for this integration. Instead, the interface types as defined in the RM-OA may be re-used. For a discussion about this integration process, see section 9.10.2.

Furthermore, in order to fill the meta-information bases on the service level, descriptive information about the source systems' functionality is extracted (manually or semi-automatically) from the source systems.

Note: RM-OA version 3 will extend the framework for ORCHESTRA Services by the inclusion of the semantic level.

9.2.2 Service Types

9.2.2.1 Overview

According to ISO 19119, a service is defined as a distinct part of the functionality that is provided by an entity through interfaces. If such a service has been being defined according to the rules of the ORCHESTRA Reference Model, it is called ORCHESTRA Service. However, the design and internal behaviour of such entities is outside the scope of the ORCHESTRA Architecture. They are conceived and identified by a designer of an OSN and are called

- ORCHESTRA Service Component when referring to the software component and
- ORCHESTRA Service Instance when referring to an instance in an OSN that has been deployed by a service provider with a dedicated identifier (see section 11.1.2), and whose operations may be called by a service consumer.

Principally, the ORCHESTRA Architecture just deals with types of ORCHESTRA Services. ORCHESTRA Service Types (short: service types) are described on a platform-neutral level in abstract service descriptions which refer to specifications of the interfaces that together provide the externally visible behaviour of the service type. In the ideal case, through a service mapping process, such a service type is mapping to respective implementation specifications for one or more given platforms. When implemented they result in ORCHESTRA Service Components and are later deployed as ORCHESTRA Service Instances in ORCHESTRA Service Networks.

Note, however, that for convenience and readability reasons the RM-OA only distinguishes between ORCHESTRA Service Types, ORCHESTRA Service Components and ORCHESTRA Service Instances when only one is meant. Otherwise, the more general term ORCHESTRA Service is used.

The conceptual schema for the specification of an ORCHESTRA Service Type is provided in the subsequent sections and illustrated in Figure 28. The main ideas are as follows:

- There is a 1:1 relationship between the abstract description of an ORCHESTRA Service Type and an ORCHESTRA Service Type. This means that each abstract service description exactly specifies one service type and vice versa.
- There is a 1:n relationship between an ORCHESTRA Service Type and an implementation specification of an ORCHESTRA Service Type. This means that each implementation specification of an ORCHESTRA Service exactly specifies one service type, and, for each service type there may be one or more corresponding implementation specifications.
- As a consequence, there is a common list of ORCHESTRA Service Types for platform-neutral and platform-specific specifications.

9.2.2.2 Platform Properties

As a general guideline, the platform shall be conformant to the OASIS Reference Model for Service Oriented Architecture 1.0 (SOA-RM, 2006). Thus, when referring in the RM-OA to characteristics of the service platform, the following terms of (SOA-RM, 2006) are used. Note that they are only pre-fixed with SOA-RM in order to distinguish them from RM-OA terms:

- SOA-RM Service: The means by which the needs of a consumer are brought together with the capabilities of a provider.
- SOA-RM Capability: A real-world effect that a service provider is able to provide to a service

consumer.

- SOA-RM Action model: The characterization of the permissible actions that may be invoked against a service.

  Note: Interacting with a service involves performing transactions with the service. Usually this is accomplished by sending and receiving messages.

- SOA-RM Service Interface: The means by which the underlying capabilities of a service are accessed.
- SOA-RM Information Model: The characterization of the information that is associated with the use of a service. Only information and data that are potentially exchanged with a service are generally included within that service's information model. The scope of the information model includes the format of information that is exchanged, the structural relationships with the exchanged information and also the definition of terms used.
- SOA-RM Execution Context: The set of technical and business elements that form a path between those with needs and those with capabilities and that permit service providers and consumers to interact.

9.2.2.3 OMM_ServiceType

The conceptual schema for the specification of ORCHESTRA Service Types is illustrated in Figure 28 (see meta-class OMM_ServiceType). The structural refinement of service types in terms of interface types is given in Figure 29 (see meta-class OMM_InterfaceType).

An ORCHESTRA Service Type is modelled by the meta-class OMM_ServiceType with the following properties:

- name: Provides the name of the service type. This name shall indicate the intended behaviour of the service type and may be used in the identification of a service type by a human user.
- abstractDesc: Association role providing a reference to the abstract description of the service type (see OMM_ServiceAbstractDesc).
- implSpec: Association role providing the list of references to service implementation specifications (see OMM_ServiceImplSpec). A reference is provided through the name of the corresponding implementation specification of the service type.
- ifName: Association role providing the list of interface types (see OMM_InterfaceType) that are supported by the service type. Interface types may be optional, i.e. all their operations are considered to be marked as <<optional>> (see section 9.2.4.3 about the meaning of optional operations).

OA_ServiceType is an instance of the meta-class OMM_ServiceType. Rules for ORCHESTRA Service Types are provided in section 9.2.5.2.

The functional classification of ORCHESTRA Service Types is described in section 9.3

9.2.3 Structure of the ORCHESTRA Service Specification Process

The structure of the specification process for ORCHESTRA Services is illustrated by the conceptual models specified in UML in Figure 28. According to the ORCHESTRA Reference Model as described in section 5.3, ORCHESTRA Service Types are specified on a platform-neutral and on a platform-specific level.

The abstract specification level is represented by the meta-classes OMM_ServiceAbstractDesc and OMM_InterfaceAbstractSpec whereas the platform level is represented by the meta-classes OMM_ServiceImplSpec, OMM_ServiceMappingSpec and OMM_PlatformSpec.

9.2.3.1 OMM_ServiceAbstractDesc

OMM_ServiceAbstractDesc represents an abstract description of an ORCHESTRA Service Type that is platform-neutral (i.e. independent of a particular service platform) and may thus be mapped to several service platforms. It provides a summary description of the functionality that the service type offers to a

calling client through its external interface. This description may be provided in different forms but in most cases comprises a human-readable text. An example for such a description is the service description framework used in the RM-OA, see section 9.4. However, the abstract description of a service is also considered to be meta-information about the service type. Thus, respective OAS-MI or parts of it may also be used as abstract service descriptions. See Annex A3 of the RM-OA for examples.

OMM_ServiceAbstractDesc has the following properties:

- serviceType: Association role providing the name of the service type that is being described.
- description: Description of the purpose and functionality provided by the service type..
- ifSpec: Association role providing the list of abstract specifications of the interfaces (OMM_InterfaceAbstractSpec) that are supported by the service type that is described in the abstract description.

9.2.3.2 OMM_InterfaceAbstractSpec

OMM_InterfaceAbstractSpec represents an abstract specification of an interface type that is platform-neutral (i.e. independent of a particular service platform). It comprises a collection of operations that together provide a self-contained set of functionality in the sense that its granularity is eligible to be reusable by other service types.

OMM_InterfaceAbstractSpec has the following properties:

- ifName: Association role providing the name of the interface type that is being specified.
- spec: Specification of the purpose and functionality of the interface type.

9.2.3.3 OMM_ServiceImplSpec

OMM_ServiceImplSpec represents an implementation specification of an ORCHESTRA Service that is specified according to the rules of a particular service platform.

- name: Name of the implementation specification of the service type.
- actionModel: Specification of the permissible actions against the service type, i.e. the SOA-RM Action Model of the service type.
- abstractDesc: Association role providing the reference to the abstract service description upon which the implementation specification is based (see OMM_ServiceAbstractDesc).
- platformSpec: Association role providing the specification of the (service) platform for which the implementation specification is valid (see OMM_PlatformSpec).
- mappingSpec: Association role providing the reference to the specification of the service mapping that links the SOA-RM Action Model of the implementation specification to the operations of the abstract service interfaces (see OMM_ServiceMappingSpec). Such a mapping specification is a mandatory part of the implementation specification of a service.

As the ORCHESTRA Architecture provides the platform-neutral view, the OMM-Service only provides detailed rules for the abstract descriptions and interface specifications of ORCHESTRA Services (see sections 9.2.5.3 and 9.2.6). However, some general rules for implementation service specifications are given in section 9.2.11.

9.2.3.4 OMM_ServiceMappingSpec

When purely applying the architectural process of ORCHESTRA, there is a service mapping process between an abstract description and an implementation specification of an ORCHESTRA Service. This process is modelled by the meta-class OMM_ServiceMappingSpec with the properties:

- spec: Specification of how to map from the abstract level to the platform.

The service mapping process shall be carried out according to the rules given in section 9.2.9. Note that one abstract description of an ORCHESTRA Service Type may be mapped to several implementation specifications because

- implementation specifications are platform-specific, i.e. for each platform there is a dedicated

implementation specification of service types, or

- the service mapping rules allow the specification of functional subsets or different concretisations of service types even for one platform.

The service mapping process also determines if an operation that is specified for a particular service type is to be called in a synchronous or in an asynchronous interaction. This is handled as part of the rules specified in section 9.2.9.

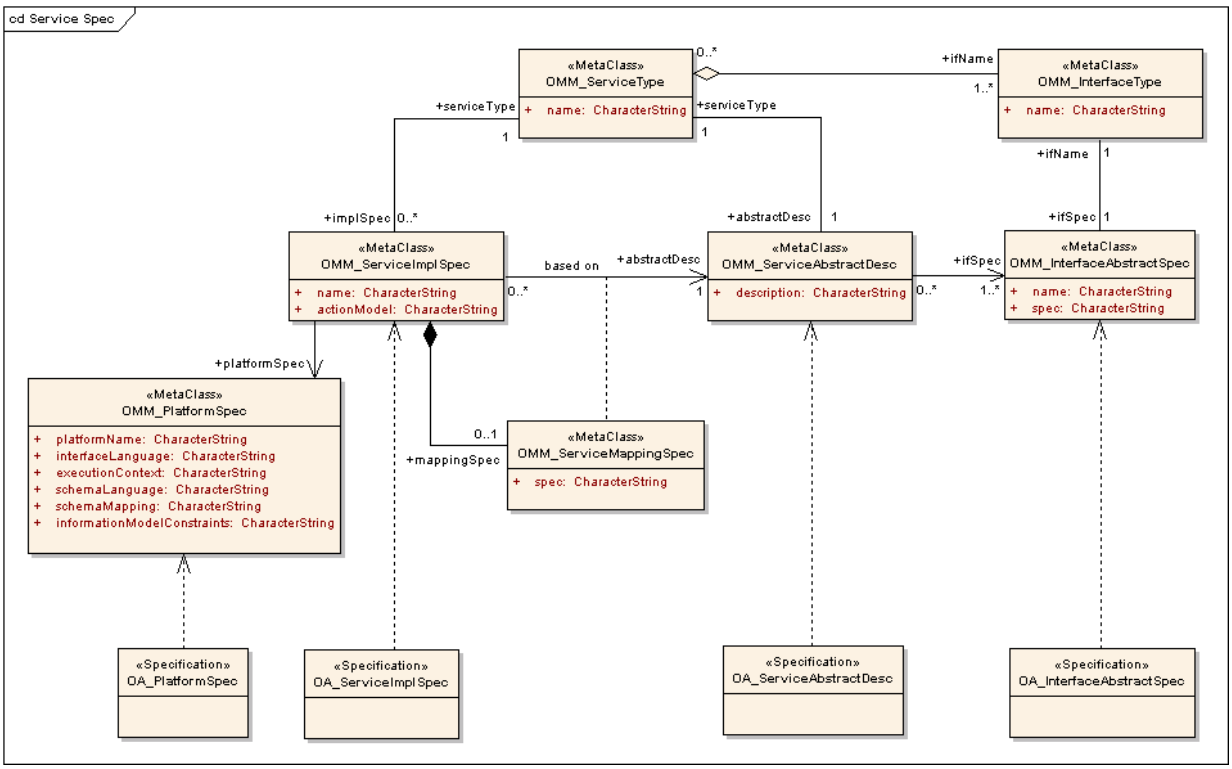

Figure 28: Specification Process for ORCHESTRA Services

9.2.3.5 OMM_PlatformSpec

The two-step mapping approach from the abstract to the implementation service specification requires that the (service) platform has been specified beforehand in a platform specification. This is modelled by the meta-class OMM_PlatformSpec in Figure 28.

The OMM_PlatformSpec provides the following properties:

- platformName: Name of the platform. In case of a standard platform, a reference shall be provided.
- interfaceLanguage: Specification of the formal language that is used to define SOA-RM Service Interfaces. In case of a standard language, a reference shall be provided.
- executionContext: Specification of the SOA-RM Execution Context. In case of a standard SOA-RM Execution Context, a reference shall be provided.
- interfaceMapping: Specification of how the interface operations on the abstract level are mapped to actions of the SOA-RM Execution Context. This specification shall cover the following aspects:
  - principle handling of synchronous and the asynchronous interactions,
  - a description of the mechanisms by which “call by value” vs. “call by reference” action parameters are supported,
  - a description of if and how optional actions and optional action parameters are sup-

ported and what optionality means for this particular platform,

- an implementation specification of the basic interface types as specified in section 9.6.1,
- an implementation specification of the way the UAA concepts (see section 7.5) are realised for the platform, e.g. how session information is handled in the interactions.

- schemaLanguage: Specification of the schema language used to define SOA-RM Information Models.
- schemaMapping: Specification of how to map from the abstract level specified in UML to the schema language used in the platform and vice-versa.
- informationModelConstraints: Specification of the constraints on the SOA-RM Information Model, especially the constraints on the format of the messages that are required to accomplish the SOA-RM Action model.

An example for a platform is the Web Service infrastructure as defined by the W3C specifications (e.g. WSDL, SOAP V1.2) together with further refinements of ORCHESTRA, e.g. the determination of GML 3.2 as schema language and, if required, a specification of a GML schema profile. The corresponding platform mapping rules of how to map from UML to GML and vice versa are given in ISO 19136 Geography Markup Language (GML).

Rules for platform specifications are provided in section 9.2.10.

9.2.4 Interface Types

9.2.4.1 OMM_InterfaceType

Each ORCHESTRA Service Type shall refer to one or more interface types and each abstract description of a service type shall refer to one or more specifications of interface types. Furthermore, each interface type shall be specified in exactly one abstract specification of an interface.

An interface type is defined as the set of operations that characterize the externally visible behaviour of an entity providing the service. The aggregation of operations in an interface type and the definition of interface types shall be for the purpose of software reusability. The specification of an interface type shall include a static portion that includes a definition of the operations. The specification of an interface type shall include a dynamic portion that includes any restrictions on the order of invocation of the operations.

An interface type is modelled by the meta-class OMM_InterfaceType with the following properties:

- name: Provides the name of the service interface.
- opName: Association role providing the list of operations (see OMM_OperationType) that are defined in the service interface.

OA_Interface is an instance of the meta-class OMM_InterfaceType. The rules for specifying interface types according to the OMM are given in section 9.2.6.

9.2.4.2 OMM_ InterfaceInheritanceRelation

Interface types may be specialised by means of inheritance. Thus, generic interface types may be defined and re-used or refined in other abstract interface specifications. This is modelled by the meta-class OMM_InterfaceInheritanceRelation.

OMM_InterfaceInheritanceRelation is the meta-class that describes a generic relationship between a more general interface type (supertype) and one specialised interface type (subtype). An interface type A being a subtype of another interface type B (that acts as supertype) supports all operations defined in B in addition to the operations defined in A. An interface type may inherit operations from more than one supertype (multiple inheritance).

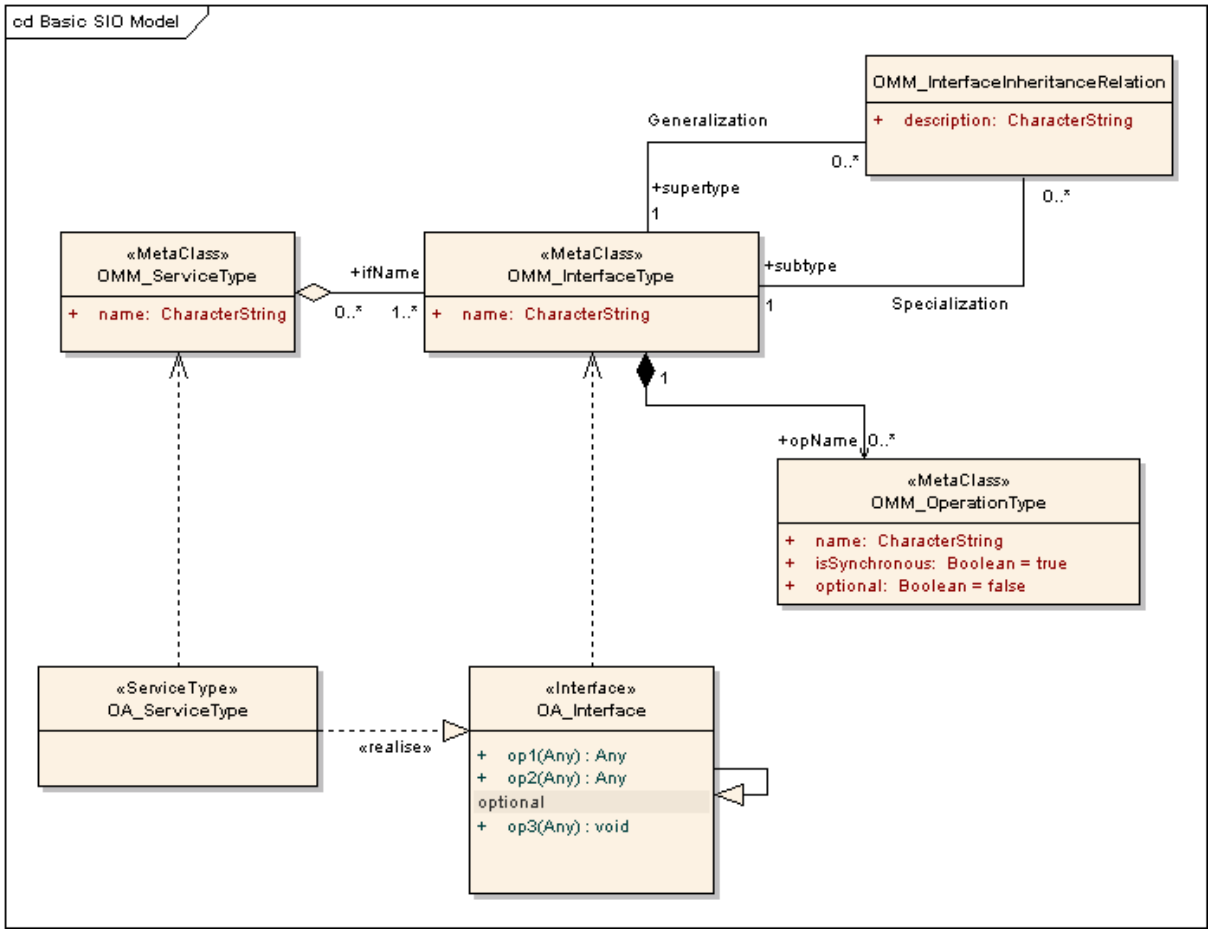

Figure 29: The Service Interface Part of the OMM

OMM_InterfaceInheritanceRelation is defined with the following properties:

- name: Name of the generalization/specialisation (optional).
- description: Explanation of the generalization/specialisation to be provided in the abstract interface specification.
- Generalization: Association specifying that an interface type has the role of being a supertype in an inheritance relationship with another interface type.
- Specialization: Association specifying that an interface type has the role of being a subtype in an inheritance relationship with another interface type.
- supertype: The role of being the more generic interface type of one other or many other interface types.
- subtype: The role of being the more specific interface type of one other or other interface types.

9.2.4.3 OMM_OperationType

The conceptual model for operations is illustrated in Figure 30. An operation type is syntactically defined through its signature that consists of the name of the operation and the request, result and exception parameters. Operations are modelled in the meta-class OMM_OperationType with the following properties:

- name: Name of the operation type.
- optional: Boolean value indicating if the operation may be omitted in the service mapping from the abstract to the implementation specification (optional = true) or if it shall be supported in the respective SOA_RM Action Model of the an implementation specification (optional = false), in the latter case either as a mandatory action or as an optional action.

- request: Association specifying that an operation type may have zero, one or more request parameter types (OMM_RequestParameterType).
- result: Association specifying that an operation type may have zero or one result parameter types (OMM_ResultParameterType).
- exception: Association specifying that an operation type may have one or more request exception parameter types (OMM_ExceptionParameterType).

Rules for operation types are provided in section 9.2.7.

All parameter types are specified as subtypes of OMM_AttributeTypes. Therefore the rules that are specified for attribute types as part of the Information Viewpoint in section 8.7 are also applied for parameter types. In fact, this means that the totality of the information exchanged in operation requests, results and exceptions is specified as an OAS. Specific rules for parameter types are provided in section 9.2.8.

9.2.4.4 OMM_RequestParameterType

OMM_RequestParameterType is a meta-class representing a parameter to be provided as part of an operation request. It has the following properties:

- name: Name of the request parameter type.
- optional: Boolean value indicating if the request parameter may be omitted in the service mapping from the abstract to the implementation specification (optional = true) or if it shall be supported in the respective operation of the an implementation specification (optional = false), in the latter case either as a mandatory parameter or as an optional parameter.

9.2.4.5 OMM_ResultParameterType

OMM_RequestParameterType is a meta-class representing a parameter to be provided as part of an operation result if the processing of the operation has been successful. It has the following properties:

- name: Name of the result parameter type.

9.2.4.6 OMM_ExceptionParameterType

OMM_ExceptionParameterType is a meta-class representing a parameter to be provided as part of an operation exception if the processing of the operation has not been successful. It has the following properties:

- name: Name of the exception parameter type.

9.2.4.7 OMM_OperationRequest

OMM_OperationRequest is a meta-class representing the set of request parameters to be provided as part of an operation call. It has the following properties:

- opName: Association role representing the name of the corresponding operation.
- paraName: Association role referring to the set of request parameters required for the operation call.

Note: The meta-class OMM_OperationRequest is required in order to model the case where all request parameters are modelled in one UML class with the individual request parameters being attributes of this class. This is, for example, required when the *SynchronousInteraction* (see section 9.6.2) or the *AsynchronousInteraction* (see section 9.6.3) interface types as specified in sections 9.6.1) are used.

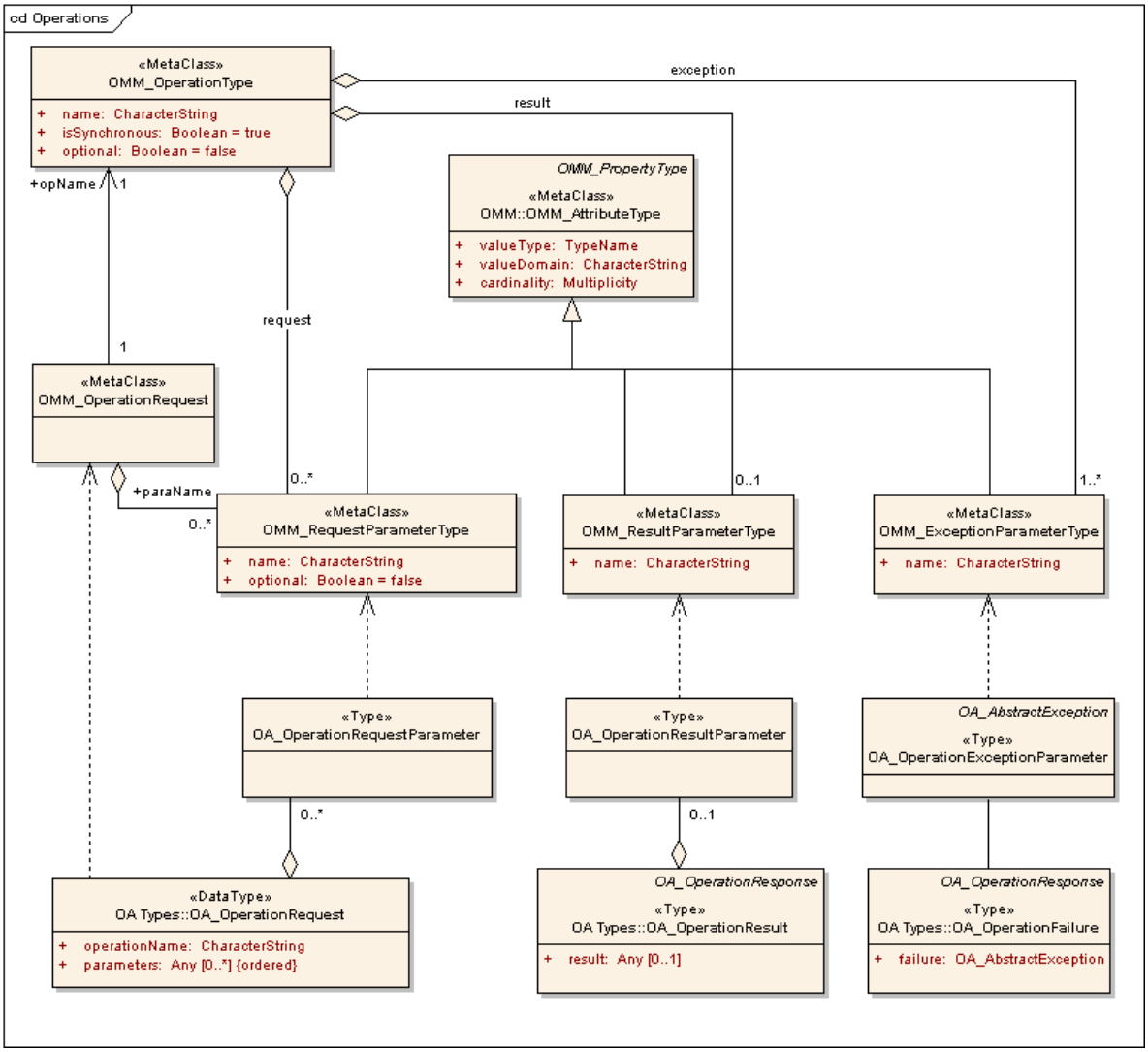

Figure 30: Model of OMM Operations and Parameter Types

9.2.5 Rules for ORCHESTRA Services

9.2.5.1 General Approach

The modelling process for ORCHESTRA Service Types shall obey the rules specified in the following sections. In this process, two cases are to be distinguished:

1. ORCHESTRA Service Types that are in a first step specified on a platform-neutral level, i.e. in addition to the mandatory abstract service description there are abstract specifications of all of their interface types and then, in a second step, are mapped to one or more platforms as specified in corresponding implementation specifications.
2. ORCHESTRA Service Types that are directly specified in an implementation specification without the delivery of abstract specifications of their SOA-RM Action Model in terms of abstract interface types in addition to the mandatory *ServiceCapabilities* interface type.

Note 1: The implementation specification is dependent on the platform specification that contains the mapping rules from and to the abstract level. Thus, it is assured that an ORCHESTRA Service Type, even when just specified on a platform level, is compliant to the OMM.

Note 2: Whether it is possible to automatically derive from a given SOA-RM Action Model of an implementation specification an abstract specification of a corresponding interface such that this distinction is not necessary will be investigated.

Rules:

1) For all ORCHESTRA Service Types an abstract description (i.e. an instance of OMM_ServiceAbstractDesc) shall be provided.

2) For all ORCHESTRA Service Types that are categorised as OA Services an abstract specification of all of their interface types (i.e. an instance of OMM_InterfaceAbstractSpec) is mandatory.

3) For all ORCHESTRA Service Types that are categorised as OT Services and thus are part of an OAA, an abstract specification of all of its interface types is optional. It is strongly recommended to provide abstract interface specifications if

   - it is envisaged to submit the service specification to a standardisation organisation that is not fixed to a particular service platform (e.g. ISO or OGC),
   - parts of the specified functionality of the service type are expected to be re-used by other service types,
   - the foreseen lifetime of the service specification is expected to be above the usual innovation cycle of IT service infrastructure technology (around 5-10 years),
   - it is envisaged to provide at least two different implementation specifications according to the same service requirements (e.g. several service profiles for the same platform or the same service profile for different platforms).

9.2.5.2 Rules for ORCHESTRA Service Types

Rules:

1) An instance of OMM_ServiceType shall be implemented as a CLASS stereotyped as <<ServiceType>> (see OA_ServiceType) that defines an ORCHESTRA Service Type as a realisation of one or more interfaces (OA_Interface). The name of the CLASS corresponds to the service type name and shall be unique for all applications of the ORCHESTRA Architecture.

   Note: Means how to assure the uniqueness of service type names will be discussed in a future version of the RM-OA.

2) An instance of OMM_ServiceType shall at least realise the interface type *ServiceCapabilities* as specified in section 9.6.1).

9.2.5.3 Rules for Abstract Descriptions of ORCHESTRA Services

Rules:

1) An instance of OMM_ServiceAbstractDesc shall be implemented as a CLASS stereotyped as <<Specification>> (see OA_ServiceAbstractDesc). It shall describe the purpose and scope of the service type in a human readable form and shall provide an overview about the interface types supported by the service type. If no other form is requested by a project environment, the RM-OA Service Description Framework as introduced in section 9.4 shall be used.

2) An instance of OMM_ServiceAbstractDesc shall refer to one or more instances of OMM_InterfaceAbstractSpec.

Note: The abstract description of an ORCHESTRA Service Type may also be combined with the abstract specification of the associated interface types (see section 9.2.6) in one “abstract service specification”. The service types that are described in the RM-OA Service Viewpoint are specified like that, see (ORCH-AbstrServ 2007).

9.2.6 Rules for the Specification of Interface Types

Rules:

1) An instance of OMM_InterfaceType shall be implemented as a CLASS stereotyped as <<Interface>> (see OA_Interface) that defines the set of operations implemented as instances of OMM_Operation.

2) An instance of OMM_InterfaceType shall be specified in UML 2.0.

3) An instance of OMM_InterfaceType (acting in the role of a subtype) may only inherit operations from those instances of OMM_InterfaceTypes (acting in the role of supertypes) if these supertypes are marked by the tagged value <<supertype>.

   Note: The supertypes need not be specified in the same abstract specification (an instance of OMM_InterfaceAbstractSpec) as the subtype.

4) An instance of OMM_InterfaceType shall be contained in exactly one abstract specification of an interface type (an instance of OMM_InterfaceAbstractSpec).

5) An instance of OMM_InterfaceAbstractSpec shall be implemented as a CLASS stereotyped as <<Specification>> (see OA_InterfaceAbstractSpec). It shall provide an overview about the interface type both in a human-readable form and in a formal specification (see rule 4) above). If no other form is requested by a project environment, the specification template applied in (ORCH-AbstrServ 2007) shall be used.

6) If an interface type contains stateful operations, i.e. if the service implementing the interface must maintain the value of a state attribute beyond the duration of the processing of an operation request, the interface specification shall contain a state diagram that describes the meaning of each state and the conditions for the transitions between the states.

9.2.7 Rules for the Specification of Operation Types

Rules:

1) An instance of OMM_OperationType shall be implemented as OPERATION of a class stereotyped as <<Interface>> (see OA_Interface) with the following properties:

   - The associated request parameters of an operation type (see instances of OMM_RequestParameterType) shall be implemented as parameter(s) of the interface operations.
   - The associated result parameters of an operation type (see instances of OMM_ResultParameterType) shall be implemented as return type of the interface operations.

2) The set of request parameters of an operation type (i.e. instances of OMM_RequestParameterType) may be summarised in one instance of OMM_OperationRequest and implemented as a CLASS stereotyped as <<Type>>. This is at least required in the following cases:
   - if the operation is to be called by means of the generic *invoke* operation of the *SynchronousInteraction* or *AsynchronousInteraction* interface type specified in section 9.6.1. See also the corresponding rules in section 9.2.9.
   - if one of the request parameters has to be specified as optional parameter (see rule 3) of section 9.2.8).

3) If an instance of OMM_OperationType may be omitted in the mapping to the SOA-RM Action model (SOA-RM 2006) of an implementation specification of an ORCHESTRA Service, the cor-

responding operation shall be marked with a stereotype <<optional>> in the class stereotyped as <<Interface>>.

Note: An instance of OMM_Operation that is not marked with a stereotype <<optional>> is considered to be a mandatory operation. This means it shall be mapped to a corresponding action in the implementation specification. This is the default case.

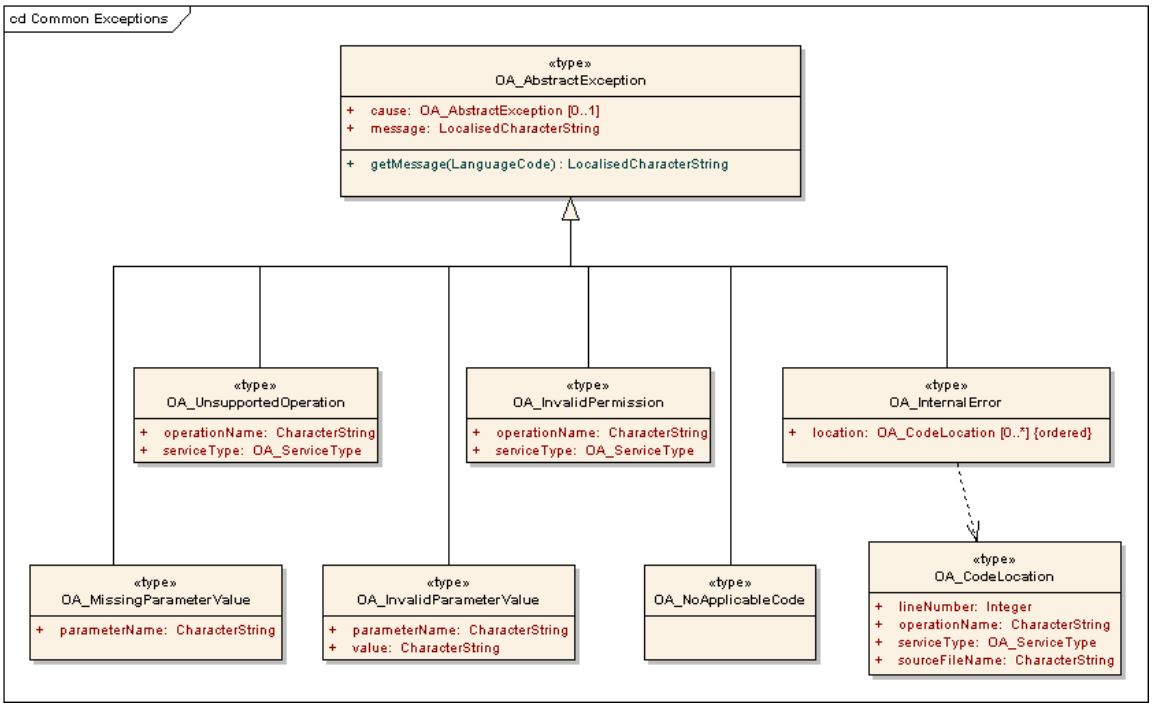

Figure 31: Specification of Exception Types

9.2.8 Rules for the Specification of Parameter Types

Rules:

1) An instance of OMM_RequestParameterType representing one request parameter of an operation shall be implemented as a CLASS stereotyped as <<Type>> (see OA_OperationRequestParameter in Figure 30).

2) An instance of the OMM_RequestParameterType shall obey the rules for the instances of OMM_AttributeTypes as specified in section 8.8.7.

   Note: This rule means that the data type of a request parameter is either a basic data type (see section 8.7.2.2) or a class with a valid stereotype (e.g., <<feature type>>). Note that this rule may cause implementation problems when applied to concrete service platforms. An example is the use of the latest GML version with the Web service development tools. In this case, exceptions from this rule must be expressed in the platfrom specification (see section 10.6)

3) If at least one instance of OMM_RequestParameterType as part of an operation type is to be specified as optional parameter, an instance of OMM_OperationRequest shall be implemented as a class stereotyped by <<DataType>> that contains all request parameters as ATTRIBUTE whereby the optional request parameters shall have the cardinality [0..1] or [0..n].

4) An instance of OMM_ResultParameterType representing a result parameter of an operation (i.e. a normal response) shall be implemented as a CLASS stereotyped as <<Type>> (see OA_OperationResultParameter in Figure 30).

5) An instance of OMM_ResultParameterType shall obey the rules for the instances of OMM_AttributeTypes as specified in section 8.8.7.

   Note: This rule means that the data type of a result parameter is either a basic data type or a class with a valid stereotype (e.g., <<feature type>>).

6) An instance of OMM_ExceptionParameterType representing an exception parameter of an operation (i.e. a failure response) shall be implemented as a CLASS stereotyped as <<Type>> (see OA_OperationExceptionParameter in Figure 30). It shall be derived from the CLASS OA_AbstractException as specified in Figure 31.

7) An instance of OMM_ExceptionParameterType shall re-use the exception types that are predefined by the specification of the exception types in UML in (ORCH-AbstrServ 2007)) if the semantics of these exception types fit the needs of the operation type.

8) An instance of OMM_OperationType together with its related instances of OMM_RequestParameterType representing an operation with its request parameters shall be implemented by a CLASS stereotyped as <<DataType> (see OA_OperationRequest in Figure 30). The operation request shall be sent either within a synchronous interaction, which is the default case, or within an asynchronous interaction.

   Note: The interfaces of a synchronous or asynchronous interaction are specified in the sections 9.6.2 and 9.6.3). Rules for their application are given in section 9.2.9.

9) An instance of OMM_ResultParameterType representing an operation result parameter shall be implemented by a CLASS stereotyped as <<Type> (see OA_OperationResult in Figure 30). The operation result is received within a synchronous or asynchronous interaction depending on the interaction mode of the preceding operation request (see rule 8) above).

10) An instance of OMM_ExceptionParameterType representing an operation exception parameter shall be implemented by a CLASS stereotyped as <<Type> (see OA_OperationFailure in Figure 30). The operation exception is received within a synchronous or asynchronous interaction depending on the interaction mode of the preceding operation request (see rule 8) above).

9.2.9 Rules for the Service Mapping to a given Platform

9.2.9.1 General Approach

The process of the service mapping to a given platform is illustrated by the conceptual model in Figure 32.

Rules:

1) For each service type that is considered to be available for a given platform an implementation specification for this platform according to rules of section 9.2.11 shall be available.

2) The process of mapping an abstract specification to an implementation specification shall be documented in a service mapping specification, i.e. an instance of OMM_ServiceMappingSpec (see rule 4) below).

3) The service mapping specification shall be a section in the ORCHESTRA Implementation Specification. Furthermore,
   - It shall define the mapping of each operation type and parameter type specified in abstract interface specifications to the SOA-RM Action Model of the ORCHESTRA service on platform level.
   - The mapping shall comprise both the static part (signature) as well as the behaviour of the operation.

Note: See (ORCH-ImplServ 2006) of an example of such a service mapping specification for the ORCHESTRA Web Services platform.

4) The service mapping specification shall consider the following cases:

- Case 1: Service Profile, an instance of OMM_ServiceProfile, if the SOA-RM Action Model of the implementation specification comprises a subset of the interface operations specified in the abstract specification of an ORCHESTRA Service Type whereby the structure and the semantics of the interface operations and the SOA-RM Action Model are identical. Rules for a Service Profile are given in section 9.2.9.2.

Note: Other cases (such as ontology-based service mediation) may be considered in future versions of the RM-OA, e.g. if the semantics of the interface operations on the abstract level and the SOA-RM Action Model on the platform level are similar but not identical.

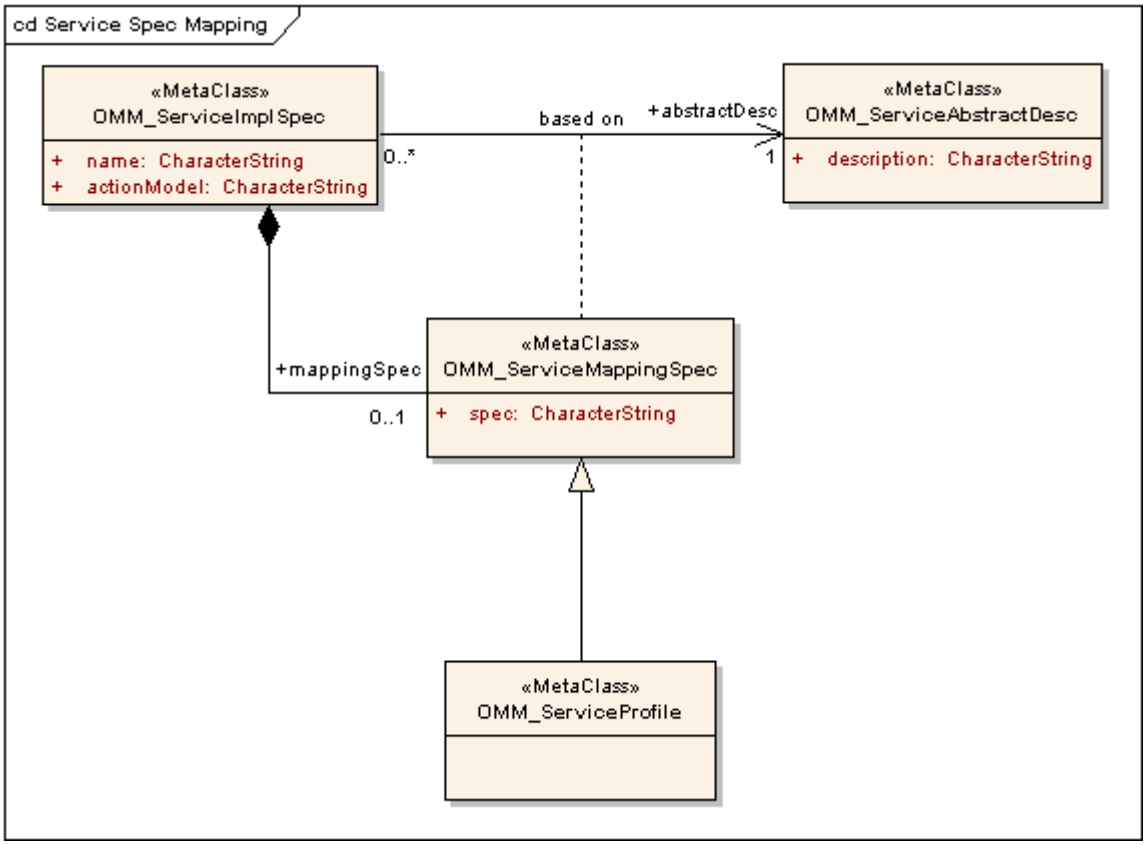

Figure 32: Structure of the Service Mapping in the OMM

9.2.9.2 Rules for Service Profiles

Rules:

1) All operations of all interfaces that are not marked as “optional” (see rule 3) of section 9.2.7) shall be mapped to an implementation specification. An operation shall be represented in the respective SOA-RM Action Model according to one of the following cases:
   - It is mapped to exactly one action invoked against a service specified in an implementation specification. The action invocation is performed in a synchronous interaction and shall be semantically identical to the operation call of the abstract specification.
   - It is mapped to the SOA-RM Action Model that provides the *SynchronousInteraction* or *AsynchronousInteraction* interface type for the given platform if the corresponding functionality has been specified for this platform (see rule 2) of section 9.2.10). In this case, the following rules apply respectively for the chosen interaction mode.

2) For all operations of all interfaces that are marked as “optional” (see rule 3) of section 9.2.7) the following cases are possible:
   - They may be omitted in the SOA-RM Action Model of the implementation specification.
   - They may be mapped to optional actions in the SOA-RM Action Model of the implementation specification.
   - They may be mapped to mandatory actions in the SOA-RM Action Model of the implementation specification.

3) A parameter of an operation that is not marked as “optional” in the abstract specification (see rule 3) of section 9.2.8) shall be syntacticly mapped to exactly one parameter of the action invocation. The parameter semantics shall be identical.

4) For all parameters of an operation that are marked as “optional” (see rule 3) of section 9.2.8) the following cases are possible:
   - They may be omitted in the action of the implementation specification.
   - They may be set to a constant value for the action in the implementation specification.
   - They may be mapped to optional action parameters in the implementation specification.
   - They may be mapped to mandatory action parameters in the implementation specification.

Note 1: The meaning of the expression “is semantically identical” is that the “real-world effect” of an action (see OASIS RM-SOA, 2005) is identical.

Note 2: It may turn out that “semantically identical” mappings are not possible in all cases and a weaker definition is required. In this case, a further case in the service mapping rules will be introduced.

9.2.10 Rules for Platform Specifications

Rules:

1) An instance of OMM_PlatformSpec shall be implemented as a CLASS stereotyped as <<Specification>> (see OA_PlatformSpec). It shall describe the basic properties of the platform as specified in section 9.2.3.5.

   Note: A more refined discussion of the platform properties is provided in the RM-OA Technology Viewpoint, see section 10.

2) An instance of OMM_PlatformSpec shall contain or refer to implementation specifications of all mandatory basic interface types specified in section 9.6.1 for which a respective functionality shall be offered for this platform. The provision of an implementation specification of the *ServiceCapabilities* interface type is mandatory.

3) An instance of OMM_PlatformSpec shall observe the conformance guidelines given in section 4 of (SOA-RM, 2006).

4) The specification of the SOA-RM Information Model constraints for platform services shall include a specification of how the rules of the OMM Service Meta-model for request, result and exception parameters (see section 9.2.8) are fulfilled. This assures that the interactions between service providers and consumers are compliant to the OMM even in cases where the interfaces to ORCHESTRA services are not first specified on an abstract level according to the OMM and then mapped to the SOA-RM action model of a particular platform.

9.2.11 Rules for Implementation Specifications of ORCHESTRA Services

Rules:

1) An ORCHESTRA Implementation Specification of an ORCHESTRA Service Type, i.e. an instance of OMM_ServiceImplSpec, shall be provided according to the rules of the chosen (service) platform (see section 9.2.10).

2) An ORCHESTRA Implementation Specification of an ORCHESTRA Service Type shall be a document that is structured according to a template that fits the chosen platform and is part of an ORCHESTRA Implementation Specification for that platform.

3) If the functionality of the ORCHESTRA Service Type has been specified in terms of abstract interface types (i.e. instances of OMM_InterfaceAbstractSpec) in addition to the mandatory *serviceCapabilities* interface type, there must be an instance of OMM_ServiceMappingSpec (see section 9.2.9) that specifies the mapping process from the abstract to the implementation specification.

9.3 Functional Classification of ORCHESTRA Services

9.3.1 Overview

As part of the ORCHESTRA Architecture, ORCHESTRA Service Types are defined by the collection of the interface types that they support. As an interface type defines the externally visible behaviour, an ORCHESTRA Service Type is in fact defined by the functionality that it provides to the external world. The RM-OA classifies service types into service categories by discussing their functionality. The main service categories are ORCHESTRA Architecture Services (OA Services) and ORCHESTRA Thematic Services (OT Services):

- An OA Service provides a generic, platform-neutral and application-domain independent functionality.
- An OT Service provides an application domain-specific functionality built on top and by usage of OA Services and/or other OT Services.

Note 1: Here and in the following, the term “usage” means that a service may call operations of another service in order to provide the desired functionality. In this sense, the calling service depends on the other service. In the service specification it is stated if such a usage is mandatory or just recommended.

Note 2: The list of OA Services and OT Services as presented in the following section is the result of an intense analysis of the functional user requirements within the ORCHESTRA project.

Note 3: The granularity for the services is oriented at the functional coherency of the service operations and the type of information (e.g. feature types, meta-information) that is managed by the service.

9.3.2 OA Services

OA Services are further classified into two sub-categories:

- OA Info-Structure Service: These are OA Services that are required to operate an OSN in the sense that these services play an indispensable role in the operation of an OSN depending on its required characteristics (see section 11.1). An example of such a role may be that at least one OSI of such a service must exist in one OSN environment (e.g. for the Catalogue Service, see section 9.7.5). Other examples are the various access services which shall be used when a feature of the respective type is accessed in an OSN (e.g. a document shall be accessed by usage of the Document Access Service, see section 9.7.3).
- OA Support Service: These are OA Services that support the provision of OA Info-Structure Service functionality (as an implementation option) or facilitate the operation of an OSN, e.g. providing an added value by combining them with the usage of OA Info-Structure Services.

These together comprise the generic information infrastructure (info-structure) of the RM-OA. The OA Services thus provide the functional basis for application domain-specific functionality. OA Services themselves do not address any specific thematic application domain, nor do they impose any structure on the OT Services.

Note that OA Services may themselves use other OA Services. Furthermore, OT Services may use both OA Info-Structure Services and OA Support Services in order to fulfil a given functionality.

This functional classification is illustrated in Figure 33.

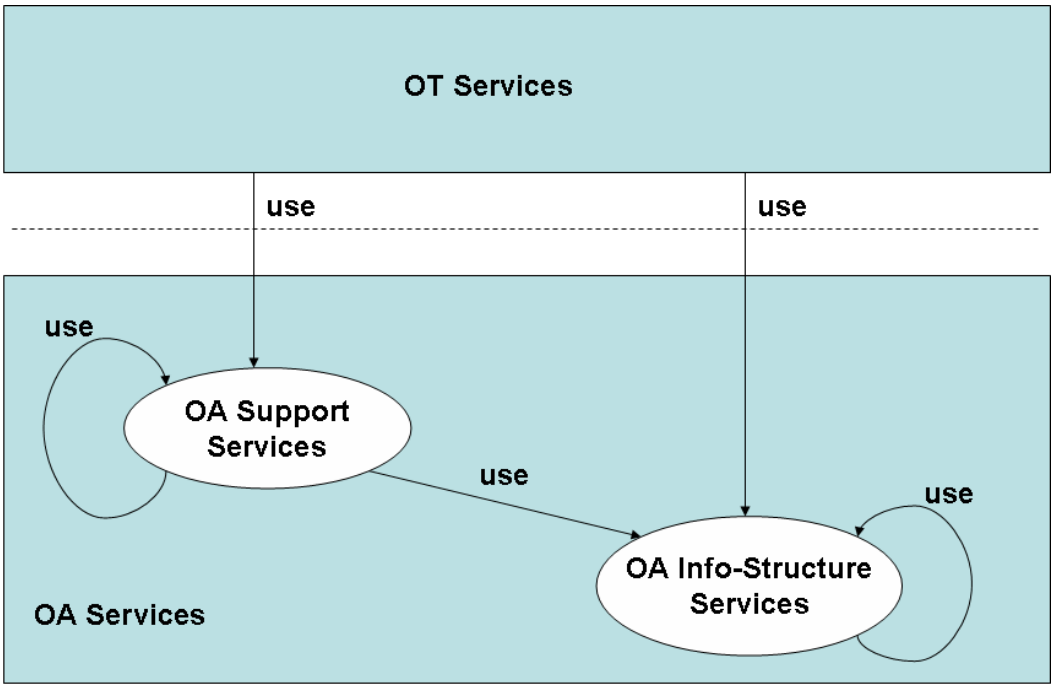

Figure 33: Functional classification of ORCHESTRA Services

Table 6 shows the current list of service types categorised as OA Services in alphabetic order within the sub-categories. Abstract specifications of these service types and its containing interface types are available in (ORCH-AbstrServ 2007).

Basic functions that may, or even shall, be offered by all OA and OT Services (e.g. an operation to retrieve a self-description of a service) with well-defined interfaces are not categorised as OA or OT Service itself. Such functions are listed in Table 5 and described as separate interface types in section 9.6. This approach follows the idea of the OMM-service (see section 7.2.3) that the "interface" is the reusable unit of specification. Note that principally, all interface types that are specified in the context of a service type may also be re-used in other service type specifications.

Interface Type Name	Section
Service Capabilities Interface	9.6.1
Synchronous Interaction Interface	9.6.2
Asynchronous Interaction Interface	9.6.3
Transaction Interface	9.6.4
Knowledge Base Interface	9.6.5

Table 5: List of Basic Interface Types

Note 1: The categorisation of an OA Service as either an OA Info-Structure service or an OA Support service is derived from the idea that essential characteristics of an OSN are discovery and access to resources residing in source systems, whereby access means read and/or write access, and, in addition, a possibility of monitoring the running services. The rationale for this selection is a compromise between, on the one hand, keeping the requirements for a service network to be "OSN-compliant" as small as possible and, on the other hand, providing a powerful service infrastructure for a broad range of ORCHESTRA Applications. In this sense, support for transformations of any kind or automatic generation of meta-information is considered to be "OA Support" as it is not required for all ORCHESTRA Applications running in a rather homogeneous environment. See a more refined discussion about OSN characteristics in section 11.1.

Note 2: The column "ISO 19119 Service Taxonomy" provides just a hint of the position of the OA Service in the ISO 19119 Service Taxonomy. Note that GeoModel/InfoManagement here stands for Geographic Model/Information Management Services.

Service Type Name	Service Category	ISO 19119 Service Taxonomy	Section
Authentication Service	OA Info-Structure	GeoModel/InfoManagement	9.7.9
Authorisation Service	OA Info-Structure	GeoModel/InfoManagement	9.7.8
Catalogue Service	OA Info-Structure	GeoModel/InfoManagement	9.7.5
Document Access Service	OA Info-Structure	GeoModel/InfoManagement	9.7.3
Feature Access Service	OA Info-Structure	GeoModel/InfoManagement	9.7.1
Map and Diagram Service	OA Info-Structure	GeoModel/InfoManagement	9.7.2
Name Service	OA Info-Structure	GeoModel/InfoManagement	9.7.6
Sensor Access Service	OA Info-Structure	GeoModel/InfoManagement	9.7.4
Service Monitoring Service	OA Info-Structure	GeoModel/InfoManagement	9.7.10
User Management Service	OA Info-Structure	GeoModel/InfoManagement	9.7.7
Annotation Service	OA Support	GeoModel/InfoManagement	9.8.3
Coordinate Operation Service	OA Support	Geographic Processing Services	9.8.1
Format Conversion Service	OA Support	GeoModel/InfoManagement	9.8.4
Gazetteer Service	OA Support	GeoModel/InfoManagement	9.8.2
Ontology Access Service	OA Support	GeoModel/InfoManagement	9.8.6
Schema Mapping Service	OA Support	GeoModel/InfoManagement	9.8.5
Service Chain Access Service	OA Support	Workflow/Task Management Services	9.8.8
Thesaurus Access Service	OA Support	GeoModel/InfoManagement	9.8.7

Table 6: List of OA Services

9.3.3 OT Services

OT Services provide application domain-specific functionality. However, both within and between different application domains, high-level functions that have a generic nature may be identified. These services are inside the scope of the RM-OA as a generic architecture and area defined as follows:

- OT Support Service: generic service that facilitates the development or interactive composition of thematic functionality.

The application domain of environmental risk management is taken as an informative example of further sub-categories of OT Services, although outside the scope of the RM-OA. Here, the ORCHESTRA project provides dedicated OT Services according to the following structure:

- OT Risk-neutral Service: service specific to the risk management domain that facilitates the development or interactive composition of risk-neutral risk management functionality.
- OT Risk-specific Service: service specific to a specific risk management domain (e.g. earthquakes, forest fires, flood, systemic risks) that facilitates the development or interactive composition of risk-specific risk management functionality.

All OT Services may use and combine the OA Services in order to fulfil their thematic function. As an example, the service sub-categories for the application domain of environmental risk management are illustrated in Figure 34.

As an example, Table 7 shows the current list of OT Support Services for the application domain of Environmental Risk Management. The column “ISO 19119 Service Taxonomy” provides a hint of the position of the OA Service in the ISO 19119 Service Taxonomy.

A candidate list of required OT Services in the domain of risk management may be found in (ORCH-D2.4.2 2005).

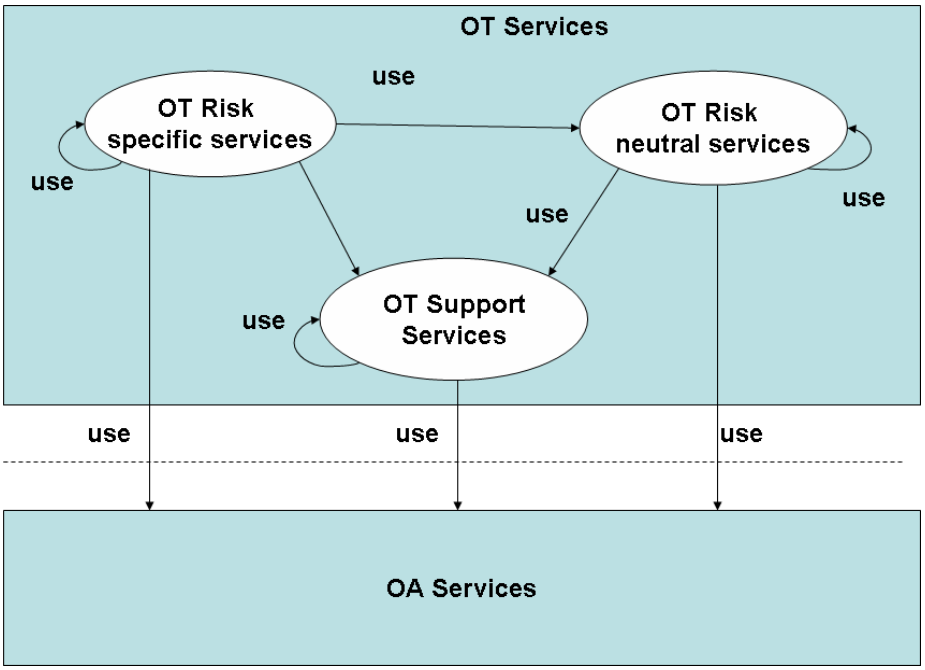

Figure 34: Example of OT Service sub-categories for the application domain of Environmental Risk Management

Note: The current list of OT Support Services is a result of functional user requirements. A subset of them (e.g. the Processing Service) is currently being specified on a detailed level. The orthers are kept for documentation and traceability purposes.

Service Name	Service Category	ISO 19119 Service Taxonomy	Section
Processing Service	OT Support	Geographic Processing Services	9.9.1
Simulation Management Services	OT Support	Geographic Processing Services	9.9.2
Calendar Service	OT Support	Workflow/Task Management Services	9.9.6
Communication Service	OT Support	Workflow/Task Management Services	9.9.5
Project Management Support Service	OT Support	Workflow/Task Management Services	9.9.4
Reporting Service	OT Support	Workflow/Task Management Services	9.9.7
Sensor Planning Service	OT Support	Workflow/Task Management Services	9.9.3

Table 7: List of OT Support Services for Environmental Risk Management

9.3.4 Human Interaction Components

The ORCHESTRA Services as categorized above do not provide an interface to a human user but rather to a software component requesting an operation at the service interface. The provision of such user interfaces is to be provided by so-called Human Interaction Components.

Human Interaction Components are software components that provide the (usually graphical) user interface (GUI) of an OA Service or OT Service. As such, the specification of such components is outside the scope of the RM-OA, i.e. no service description will be provided.

9.4 Relationship of the ORCHESTRA Service Types to INSPIRE

The ORCHESTRA Architecture follows an iterative design approach. The major iteration cycles that are currently foreseen are described in section 6.2.3. The focus of the current version 2 of the OA is to support syntactic interoperability, in particular but not exclusively for spatial services, such that the OA may contribute to the specification of the INSPIRE network services as outlined in section 6.2.2.3.

The following table provides an overview of which of the ORCHESTRA Interface and Service Types may contribute to which INSPIRE network services. This linkage to the INSPIRE requirements is preliminary as the work of the INSPIRE drafting team for network services has not yet been finalised and a detailed definition on the INSPIRE Network Services is not yet available.

INSPIRE Network Services	ORCHESTRA Interface Type	Specified in ORCHESTRA Service Type	Comment
Discovery Services	CatalogueSearchInterface	Catalogue Service (see section 9.7.5)	The ORCHESTRA Catalogue Service is generic w.r.t. the usage of a specific meta-information model. The CS-W 2.0 ISO AP 19115/19119 as currently investigated by INSPIRE could be chosen as one example.
Upload Services	CataloguePublication and CatalogueCollection Interface	Catalogue Service (see section 9.7.5)	
View Services	MapService	Map and Diagram Service (see section 9.7.2)	INSPIRE just requires rendering in maps
Download Services	FeatureAccessService	Feature Access Service (see section 9.7.1)	To support the download of feature instances
	DocumentAccess	Document Access Service (see section 9.7.3)	To support the download of predefined datasets
Transformation Services	CoordinateOperation	Coordinate Operation Service (see section 9.8.1)	
	SchemaMapping SchemaMappingRepository	Schema Mapping Service (see section 9.8.5)	In case schema mapping remains in the scope of the INSPIRE Transformation Services.
"Invoke spatial data services" services	ProcessingService	Processing Service (see section 9.9.1)	OMM-Service (see section 9.2) may provide input to the specification of the INSPIRE service reference model mentioned in the INSPIRE description
	ServiceChainAccessService	Service Chain Access Service (see section 9.8.8)	

Table 8: Possible Contribution of ORCHESTRA Service Types to INSPIRE Network Services

9.5 Service and Interface Description Framework

A coarse description of the ORCHESTRA Interfaces and Services is provided in a textual format according to the following template. The detailed abstract specifications of the services are provided in (ORCH-AbstrServ 2007). These documents contain formal specification of the information objects that are referred to in the interface operations (e.g. parameter types).

Name	Name of the ORCHESTRA Service or Interface Type Convention: All individual words in the service type name are capitalized.
Standard Specifications	Reference to an abstract or a platform-specific service specification according to a standardisation organisation (e.g. ISO, CEN, W3C, OGC,…) or to important reference material that has been taken into account when describing the service, its interfaces or operations. In case there is no adequate reference the field is set to “no corresponding standard known”
Description	Human understandable description of the functionality provided by the ORCHESTRA service or interface. The end of a service description shall provide the following text: The <*name*> Service provides its functionality through the following interfaces: • *Interface1*: human understandable description of the purpose of interface 1 • … • *InterfaceN*: human understandable description of the purpose of interface N Note: If an interface is re-used from another ORCHESTRA Service Type description, the name of this service type shall be indicated in brackets in the interface definition below. The description of the used interface operations shall be adapted to the context of the using service. Convention: All words in the interface name are written together in italics without a blank in between. The first letter of the first word and all other words are written in upper case letters.
Interface *Interface1 (from << Name of an ORCHESTRA Service>*	
oper1	Human understandable description of the operation 1 of the interface. Only major input and output information shall be described, no individual request and result parameters. Note: All words in the service operation name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case.
…	
operN (optional)	Human understandable description of the operation n of the interface. Optional operations are to be marked by suffix (optional) after the operation name.
…	
Interface *InterfaceN*	
…	
Example usage	Description of an example usage scenario of the service, e.g. by the combination of several operation calls of the service or in combination with another ORCHESTRA Service.
Comments	Description of current restrictions or possible extensions and enhancements in future versions of the RM-OA.

Table 9: Service Description Framework

9.6 Basic Interface Descriptions

9.6.1 Service Capabilities Interface

Name	Service Capabilities Interface
Standard Specifications	The *getCapabilities* operation of the Service Capabilities Interface is designed such that it is backward compatible with the concepts and definitions of the *GetCapabilities* operation as defined in • OGC 05-008c1 Web Services Common Specification V1.0 The idea is that the usage of the meta-information schema defined in that OGC standard is just one possibility how the service capabilities may look like.
Description	The Service Capabilities Interface defines of a uniform way to get a self-description of an OSI by means of so-called capabilities. The capabilities form service meta-information which can be used for various purposes like, for example, service discovery and service invocation. This *ServiceCapabilities* interface is a mandatory interface and shall be implemented by all ORCHESTRA Services.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the capabilities of an OSI. This operation takes into account that in addition to capabilities that are common to all ORCHESTRA Services (referred to as common capabilities) an ORCHESTRA Service may provide a specific set of capabilities (referred to as specific capabilities). Furthermore, this operation allows the capabilities to be delivered according to different service meta-information schemas. The meta-information schema shall be structured according to the rules for ORCHESTRA application schemas defined in section 8.8.
Example usage	The Service Capabilities Interface contributes to a consistent description of the functionality provided by ORCHESTRA Services. Thus, it helps in developing generic applications and in defining a common framework for service discovery and access.
Comments	The contents of the service meta-information are defined as part of the specification of the OAS-MI for services in Annex B1 of the RM-OA. Furthermore, the abstract specification of the Service Capabilities interface (see (ORCH-AbstrServ 2007) also contains the specification of common exception types to be used by all other ORCHESTRA interface types.

Table 10: Description of the Service Capabilities Interface

9.6.2 Synchronous Interaction Interface

Name	Synchronous Interaction Interface
Standard Specifications	No corresponding standard known.
Description	The Synchronous Interaction Interface defines of a uniform way to request synchronous execution of a service operation. Synchronous execution of an operation means that the client requests operation execution and then waits until the operation provider has finished operation execution and returns a response. Such a response may either contain an operation result value (which also may be empty) or may be an indication of a failure which is modeled as exception.
Interface *SynchronousInteraction*	

invoke	Executes an operation synchronously and returns the operation response.
Example usage	The synchronous interaction interface may be used by generic software frameworks that support the integration of source systems into an OSN (see section 9.10.2).
Comments	none

Table 11: Description of the Synchronous Interaction Interface

9.6.3 Asynchronous Interaction Interface

Name	Synchronous Interaction Interface
Standard Specifications	The following WC3 standard provides transport-neutral mechanisms to address Web services and messages. • W3C Web Services Addressing V1.0 Core, http://www.w3.org/TR/ws-addr-core/ On the abstract, platform-independent level of the service specification, the usage of WS-Addressing is out of scope. However, it has to be taken into account for the implementation specification of the Asynchronous Interaction Interface. The concept of an Endpoint Reference (EPR) as defined by WS-Addressing can be used for the *invokeAsync* operation in order to specify the entity to which notifications are to be sent as a result of asynchronous operation execution. • OASIS Web Services Notification (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn) OASIS WS-Notification is a family of related specifications that define a standard Web services approach to notification using a topic-based publish/subscribe pattern. A Web service (NotificationProducer) disseminates information (notifications) to a set of other Web services (NotificationConsumers) each of which has been subscribed to the producer previously. WS-Notification itself makes use of WS-Addressing as defined by W3C to indicate endpoint references.
Description	The Asynchronous Interaction Interface defines a uniform way to request asynchronous execution of a service operation, e.g., for operations which are time-consuming or deliver results periodically. Asynchronous execution of an operation means that the client requests operation execution but does not wait until the operation has finished. Instead, the client may execute other tasks while the operation is running. However, in most cases the client wants to be notified when the operation terminates in order to get its results. In addition, when executing an operation asynchronously the client should be able to abort operation execution.
Interface *AsynchronousInteraction*	
invokeAsync	Starts asynchronous execution of an operation. The *invokeAsync* operation returns immediately with an identifier (invocation ID) representing the asynchronous execution. In order to receive notifications a reference to a callback interface can be provided.
abort	Aborts execution of a previously invoked asynchronous operation identified by its invocation ID.
notify	Passes a notification to the callback interface provider.
Example usage	The Asynchronous Interaction Interface may be used for processing service operations (e.g. geostatistical interpolations) that take a significant time to produce results.
Comments	The objective of the Asynchronous Interaction Interface is to define a uniform way to request for asynchronous execution of a service operation. The interface can be

	implemented by a service in order to offer asynchronous execution of certain service operations. In contrast to WS-Notification, the interface does not claim to define a general publish/subscribe pattern. Both specifications are not directly related and are not comparable with respect to compatibility. The Asynchronous Interaction Interface comes along with a Notification Callback Interface which expresses the need to receive notifications in the context of asynchronous operation execution. A notification may signal the operation result or operation progress. This is a special case of notification in the sense of WS-Notification: By invoking an operation asynchronously, the caller implicitly subscribes at the operation provider. When receiving the final notification from the operation provider, the receiver implicitly unsubscribes. From that point of view, this is a special case of the publish/subscribe pattern.

Table 12: Description of the Asynchronous Interaction Interface

9.6.4 Transaction Interface

Name	Transaction Interface
Standard Specifications	• OASIS Business Transaction Protocol (BTP) 1.0, Committee Specification (http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf)
Description	The Transaction Interface supports to enclose a series of service operations in transactional brackets. In a system that supports multiple users, synchronization of access to resources must be assured. This is an especially important requirement in the context of changing resources (write access), otherwise the consistency of the state of the system and its data cannot be guaranteed. Obviously not all services need to support transactions but if they do care must be taken. In order to guarantee a great amount of flexibility, the *TransactionInterface* allows numerous different types of transactions, e.g. transactions that support the properties of atomicity, consistency, isolation, and durability (ACID), OASIS business transactions that relax some of the ACID properties, operation batching, 'best try' transactions and sub-transactions.
Interface *TransactionInterface*	
createAcid Transaction	Creates a new ACID transaction at the service
create Business Transaction	Creates a new business transaction at the service
createSubAcid Transaction	Creates a new sub ACID transaction at the service.
createSub Business Transaction	Creates a new sub business transaction at the service.
setImplicit Commit	Sets the implicit timeout action for the specified transaction.
setRollback OnFailure	Sets the default failure action for the specified transaction
setLockOwner	Sets the resource lock owner for resources allocated by this transaction.
start	Starts an existing transaction at the service

Transaction	
tryCommit	Tries to commit the transaction without rolling back if the commit failed.
commit Transaction	Makes all changes made during the transaction permanent. Also releases all locks that have been acquired (if any) during the transaction.
abort Transaction	Revokes all chances made during the transaction
suspend Transaction	Suspends the transaction environment. All operations that are invoked at the service are carried out outside the transaction environment. This does not free any acquired locks.
resume Transaction	Set the specified transaction as the currently active transaction. This does not free any acquired locks.
getActive Transaction	Retrieves the transaction ID of the (most inner, if sub transactions are supported) currently active transaction.
add Transactions	Adds a number of transactions as children to the specified transaction.
remove Transactions	Removes a number of child-transactions from the specified transaction.
Example usage	The Transactional Interface may be used when a sequence a *setFeatures* operation calls has to be carried out in an atomic fashion.
Comments	none

Table 13: Description of the Transaction Interface

9.6.5 Knowledge Base Interface

Name	Knowledge Base Interface
Standard Specifications	• W3C RDF-Schema http://www.w3.org/TR/rdf-schema/ • W3C RDF/XML Syntax Specification (Revised) http://www.w3.org/TR/rdf-syntax-grammar/ • W3C SPARQL Query Language for RDF (Candidate Recommendation) http://www.w3.org/TR/rdf-sparql-query/
Description	The Knowledge Base Interface provides access to a knowledge base in an OSN. The knowledge base can store identifiable units of knowledge, in the sequel referred to as “models”. A model has a uniform resource identifier (URI). The Knowledge Base Interface conveys query requests to models received via the OSN to the knowledge base’s local processing engine and returns the results to the OSI that requested them. The Knowledge Base Interface abstracts from existing languages for knowledge representation and querying, but it assumes that some concepts are common to most of them: • Knowledge is represented as a graph, i.e. a number of nodes and edges. • The knowledge graph is divided into a number of sub-graphs, so called “models”. • Models are described by a number of basic elements constituting the model graph; these elements describe the nodes and the edges. Updates of a model can be performed by adding/deleting basic elements. RDF is an example for a standard which fulfils these assumptions. In RDF, for in-

	stance, “statements” are the basic elements. SPARQL is a query language for RDF models. The SPARQL Protocol uses WSDL 2.0 to describe a means for conveying SPARQL queries to a SPARQL query processing service and returning the query results to the entity that requested them. The Knowledge Base Interface can partly be implemented by means of RDF storage and SPARQL queries, but other implementations are possible. The Knowledge Base Interface provides operations to query and update models contained in the knowledge base. Queries are to be formulated in a query language that is compatible with the queried model. As opposed to the Feature Access Service, the result of such a request does not necessarily need to be a feature set: the service may deliver results of any format, from complete models down to boolean values. Update requests to a model contain the new elements, which are to be added to the model, and the elements to be deleted.
Interface *KnowledgeBase*	
queryModel	Submits a query to a model stored in the knowledge base. The model to which the request is to be sent is referenced by a URI. The query is formulated in a query language which must be compatible with the knowledge representation model used by the knowledge base. The service conveys the request to the knowledge base, which executes the query and composes the result in the required result format (parameter resultFormat). If the resultFormat parameter is not present, the result is delivered in a default format.
updateModel	Submits an update request to a model stored in the knowledge base. The model to which the request is to be sent is referenced by a URI. The request contains the set of basic elements to be added and the set of elements to be deleted. The service conveys the request to the knowledge base, which executes the update request.
Interface *TransactionInterface*	
	The operations of the *TransactionInterface* are used when a synchronised access to the knowledge base must be assured, especially in the case of the *updateModel* operation of the *KnowledgeBase* interface.
Example usage	Pre-population and automatic population: In a scenario, the knowledge base can hold so-called “named entity” definitions (e.g. mountains, rivers) and relationships between them. A named entity can be inserted into the knowledge base in two ways: • Pre-population – the named entities are imported or acquired otherwise from trusted sources. • Automatic discovery and population – the named entities are discovered in the process of automatic semantic annotation (or by usage of other knowledge discovery and acquisition methods) and are then populated into the knowledge base by means of the *updateModel* operation.
Comments	The main difference between a knowledge base approach and conventional *SQL databases* is that a knowledge base is more flexible: models can be added or removed during run time and there is no fixed database schema. A knowledge base can have a schema defined by means of ontology (e.g. RDF-Schema or OWL as schema of an RDF knowledge base), but it does not necessarily need one. In its current specification, the Knowledge Base interface provides means for model update, but it does not provide means for adding and removing complete models. It is assumed that these tasks are performed via local, non-ORCHESTRA interfaces of the knowledge base (e.g. import). Nevertheless, implementation should allow adding and removing new models dynamically at runtime.

Table 14: Description of the Knowledge Base Interface

9.7 OA Info-Structure Service Descriptions

9.7.1 Feature Access Service

Name	Feature Access Service
Standard Specifications	The functionality of the Feature Access Service is based on the WFS and WCS OGC implementation specifications: • OGC Web Feature Service (WFS) Implementation Specification (latest version V1.1, 04-094) • OGC Web Coverage Service (WCS) Implementation Specification (latest version V1.1.0, 06-083r8) These specifications allows for retrieval of features and coverages, respectively. Coverages and features are considered as ORCHESTRA features at the abstract level, and thus one interface has been developed for the access to both types. The write functionalities of the WFS specifications (which basically consist of a transactional operation) have been transferred into three operations *setFeatures*, *createFeatures*, and *deleteFeatures*, as to follow the ORCHESTRA convention of operation functionality. Additionally, the objective was to put the "write behaviour" of the WFS at the operation level in the interface. Currently, in the OGC WFS specification, the write type of a given operation (i.e., insert, update, or delete) is specified as a parameter to a more generic operation (transaction operation). The lock mechanism offered by the WFS getFeatureWithLock and lockFeature must be implemented using the transaction interface offered by ORCHESTRA (see section 9.6.1). This approach ensures the same transactional model throughout all services within ORCHESTRA where (serializable) transactions are required. Finally, the *setFeatureTypes*, *createFeatureTypes*, and *deleteFeatureTypes* operations have been specified in addition to the OGC specifications in order to provide an interface to manage feature types. This is currently not possible via implementations following the OGC specifications. As the Feature Access Service does not define a specific query language or encoding for features, it is up to the implementation specification to define these. Prominent standards which can be used for query languages are: • ISO/IEC 9075:1995 Information technology -- Database languages – SQL • OGC 04-095 Filter Encoding Implementation Specification V1.1 A commonly used standard for the encoding of (especially geographic) features is: • ISO 19136 Geographic information -- Geography Markup Language (GML) GML is based on the XML standard, which can be used for encoding as well: • W3C - Extensible Markup Language (XML) 1.0 (http://www.w3.org/TR/2006/REC-xml-20060816) Examples of commonly used encodings for coverage features are: • GeoTIFF (http://www.remotesensing.org/geotiff/geotiff. html) • HDF-EOS (http://www.hdfeos.org) • CF-NetCDF (http://www.cgd.ucar.edu/cms/eaton/cf-metadata)
Description	The Feature Access Service allows interoperable read and write access on feature instances available in an OSN. Furthermore, the Feature Access Service provides

	an interface that may be inherited by more specific access services (e.g., sensor access service) using interface inheritance. The Feature Access Service offers information about: • The feature types it is capable to provide. • The supported encoding(s) to transfer requested or submitted feature data. • The query language and mechanism for filtered feature access. Features provided by the Feature Access Service are instances of a certain feature type defined in an ORCHESTRA Application Schema (OAS), which again is an instantiation of an OMM_FeatureType (see section 8.7.2). This means that the Feature Access Service only permits access to information which is represented through feature types according to the rules of the ORCHESTRA Meta-Model (OMM). Whether information is remodelled on-the-fly by a software component or whether the features are actually stored in a feature store is not crucial for the Feature Access Service. Seen from the interface, the feature representation is a black box and is not visible for clients. The Feature Access Service allows queries to select certain features based on their type, certain attribute values and their spatial and temporal extent. The selection statement is encoded using a query language that supports all these functionalities (e.g., SQL including spatio-temporal statements). By selecting and retrieving features, access to their attributes and operations is provided. Any Feature Access Service (and its possible profiles or possible inheriting interfaces) may support the update of existing feature instances, the creation of new feature and the deletion of existing features, and hence, in this case, it should also be transactional. It can also allow the creation, updates, and deletions of feature types. Feature instances and feature types are identifiable by a Unique Identifier (UID) that is unique with respect to at least one OSN (section 11.1.2). If a Feature Access Service is used to create a new feature instance or feature types it will also create an appropriate UID for this feature type or instance. Additionally, it is important to emphasize the requirements for Authorisation and authentication in order to support creation, deletion, and modification of feature and feature types (see section 7.5). The Feature Access Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *FeatureAccessService*: selection, creation, update and deletion of feature instances and feature types.
Interface *ServiceCapabilities*	
get Capabilities	Informs the requestor about the common and specific capabilities of a Feature Access Service instance. Examples of specific capabilities are the supported feature types, the encoding of feature type requests, the encoding of returned feature collections as well as the supported query language.
Interface *FeatureAccessService*	
getFeature Types	Gets a description (the schema) of given feature types serviced by an Feature Access Service instance in a specific encoding based on a query.
setFeature Types	Updates existing Feature Types matching a given query.
createFeature Types	Creates new Feature Types based on feature type descriptions.
deleteFeature Types	Deletes existing Feature Types matching a given query.

getFeatures	Retrieves features and their attributes matching a given query.
setFeatures	Updates existing features matching a given query.
createFeatures	Creates new features based on a feature collection and a given query.
deleteFeatures	Deletes existing features matching a given query.
Example usage	A client accessing this service wants to retrieve all feature instances of roads for a particular region. The Feature Access Service is passed a *getFeatures* request for the specified area and feature type. A response is generated containing all valid features. The features may be modified and submitted to the Feature Access Service as an update transaction (via the *setFeatures* operation).
Comments	As the RM-OA, in accordance with ISO 19123, considers coverages as subtypes of features, the Feature Access Service can also be used to access coverages.

Table 15: Description of the Feature Access Service

9.7.2 Map and Diagram Service

Name	Map and Diagram Service
Standard Specifications	The Map and Diagram Service is a functional extension of the following standards: • ISO/DIS 19128:2005 - Geographic information -- Web Map Server Interface • OGC 06-042 Web Map Service (WMS) Implementation Specification V1.3.0 The extensions refer to the generation of diagrams, legends, the detailed layer descriptions that are needed for fine-grained user-styling, and the management of layers and styles. When being mapped to a W3C Web Service platform, the Map and Diagram Service supports the following standards: Data sent to the Map and Diagram Service may be structured according to: • ISO 19136 Geographic information -- Geography Markup Language (GML) An alternate data source may be a feature store that provides feature instances according to: • OGC 04-094 Web Feature Service (WFS) Implementation Specification) V1.1 The following standards are used for the symbology definition: • OGC 02-070 Styled Layer Descriptor (SLD) Implementation Specification V1.0 • OGC 04-095 Filter Encoding Implementation Specification V1.1 These are extended with symbolizers for diagrams.
Description	The Map and Diagram Service is a service that visualizes, symbolizes and enables geographic clients to interactively visualise geographic and statistical data. Its main task is to transforms geographic data (vector or raster) and/or numerical tabular data (e.g. census data, result of a statistical analysis) into a graphical representation using symbolization rules. The main output of this service is an image document, which can be either in raster (e.g. jpeg, png) or symbolized-vector format (e.g. SVG). The meaning of the image document (the output of this service) is a general reference map (visualization of geographic information), a diagram (visualization of statistical data) or a thematic map (visualization of the spatial distribution of one or more statistical data themes). This service enables the integration of extended Style Layer Descriptor (SLD)

	documents, which allows the definition of symbologies and symbolization rules at the feature level and allows also the integration of user data and remotely available data from other OA Services like the Feature Access Service (see section 9.7.1) The Map and Diagram Service provides the functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *MapDiagramService*: This interface allows a client to request and receive maps, diagrams and, optionally, information about the visualized features according to specifications, as well as to put/remove data and styles on the server for visualization.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the capabilities of a Map and Diagram Service instance. Examples of specific capabilities are a document containing, among others, a list of supported operations and predefined data layers available on the server with the corresponding layer information.
Interface *MapDiagramService*	
getMap	Returns a map of spatially referenced geographic and thematic information as an image document with the characteristics specified by the client application. The characteristics of the output image are specified by the *outputAttributes* parameter (image format, width, height, transparency, etc…) as well as the *mapAttributes* parameter (list of layers and their corresponding styles, coordinate reference system, global bounding box). Optionally, the map parameters can be provided using an SLD document.
getDiagram (optional)	Returns a diagram representation of numerical data as an image document with the characteristics specified by the client application. The characteristics of the output image are specified by the *outputAttributes* parameter (image format, width, height, transparency, etc…) as well as the *diagramAttributes* parameter (list of tabular data layers and their corresponding styles – diagram type, diagram characteristics). Optionally, the diagram parameters can be provided using an SLD document. This operation expects that the data to be rendered is in tabular format.
getLayerDescription (optional)	Returns a layer description document containing schema information for a layer: attribute names, types, units, statistical information when applicable (like value ranges, max, min etc.). This information is needed by clients in order to create their own styles and symbolization rules based on attribute values.
getLayerLegend (optional)	Returns a legend symbol (corresponding to a layer) as an image document with the characteristics specified by the client application. The characteristics of the output image are specified by the *outputAttributes* parameter (image format, width, height, transparency, etc…) as well as the *styledLayer* parameter (name of the layer for which the legend should be generated and its corresponding styles). If the styles corresponding to the layer are not available on the server, then the styles have to be defined and sent again by the client (optionally, also as a SLD document).
getFeatureInfo (optional)	Returns information about the features rendered in a certain point of a map or diagram layer as a document. The request must specify the attributes of the query point (x and y coordinates of the point in the image coordinate system, the layer name, and the number of features for which is expected to receive information) as well as a copy of the request that generated the image.
setLayer (optional)	Stores a new data layer on the server if the format of the sent layer data is supported (the supported formats for data input are advertised in the service capabilities). For this operation the following information must be defined: the layer (name, data, data format, minimum and maximum scale, etc…), the duration for which the layer will be stored and also if it will be visible or not for other users. The operation confirms the success of the request by sending back to the client a Boolean

	"TRUE".
deleteLayer (optional)	Removes an existing data layer from the server. The operation confirms the success of the request by sending back to the client a Boolean "TRUE".
setStyle (optional)	Stores a new style layer on the server. For this operation the style must be defined either by sending the symbology or by referencing a remotely available symbology. Furthermore, the duration for which the style will be stored and also if it will be visible or not for other users must be defined. The operation confirms the success of the request by sending back to the client a Boolean "TRUE".
deleteStyle (optional)	Removes an existing style from the server. The operation confirms the success of the request by sending back to the client a Boolean "TRUE".
Example usage	A requestor accessing this service wants to create a map that shows the spatial distribution of the forest fire hazard zones (classified by the susceptibility level) with different colours. On top of this layer the requestor is interested to have the road network, the hydrological network, the urban areas and a diagram layer with bar charts showing the number of historical forest fire cases. The hazard zones and the historical forest fire data are accessible by means of a Feature Access Service and other layers are available on the server. The requestor now invokes a *getMap* operation by passing a styled layer descriptor document, which defines the location of the data and the symbolization corresponding for each layer. The response of the service will be a map provided in the requested format.
Comments	It is beyond of the scope of this service to provide a human interface like the geographic viewer in the human interaction services. On the other side, other map service instances, a geographic viewer or even a Web browser could act as a client to this service.

Table 16: Description of the Map and Diagram Service

9.7.3 Document Access Service

Name	Document Access Service
Standard Specifications	no corresponding standard known
Description	The Document Access Service supports access to documents of any type (textual documents, images,). A document descriptor (see section 8.7.5.2) is regarded as a specific kind of a feature type, therefore the Document Access Service is a specialisation of the Feature Access Service (see section 9.7.1) which inherits only feature-specific operations. Operations that manipulate feature types are not supported by this service, since the only feature type this service supports is OA_DocumentDescriptor. Compared with the Feature Access Service this service enables the conversion of documents and it guarantees that the returned feature instances are of type OA_DocumentDescriptor. The Document Access Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *DocumentAccessService*: Selection, creation, update and deletion of documents.
Interface *ServiceCapabilities*	

get Capabilites	Informs the client about the common and specific capabilities of a Document Access Service OSI. Examples of specific capabilities: a) the specific capabilities inherited from the Feature Access Service, b) information about supported document-encodings and MIME types.
Interface *DocumentAccess*	
get Documents	Returns and optionally converts documents. This operation is an extension of the *getFeatures* operation of the *FeatureAccessService* interface. In addition to the *getFeatures* operation it supports the conversion of a document. The *getDocuments* operation retrieves features of the feature type OA_DocumentDescriptor. A query can be specified to retrieve certain documents that meet specific requirements.
create Documents	Creates new documents of type OA_DocumentDescriptor. This method is an extension of the *createFeatures* operation of the *FeatureAccessService* interface. Since this operation provides no additional functionality, the detailed abstract specification is omitted.
set Documents	Updates existing documents. This method is an extension of the *setFeatures* operation of the *FeatureAccessService* interface. Since this operation provides no additional functionality, the detailed abstract specification is omitted.
delete Documents	Removes existing documents. A query identifies which document to be deleted. This method is an extension of the *deleteFeatures* operation of the *FeatureAccessService* interface. Since this operation provides no additional functionality, the detailed abstract specification is omitted.
Example usage	After a search in a catalogue-service a found document can be retrieved by call of the *getDocuments* operation.
Comments	None.

Table 17: Description of the Document Access Service

9.7.4 Sensor Access Service

Name	Sensor Access Service
Standard Specifications	• OGC 06-009r1 – Sensor Observation Service Implementation Specification V0.1.5 (Request for Comments) • OGC 05-086r2 - Sensor Model Language (SensorML) Implementation Specification V1.0 (Draft proposed version)
Description	This service provides a basic interface for accessing sensor data, configuring a sensor and publishing sensor data. While the configuration and data publishing interfaces of the Sensor Access Service are optional, the ability to find a certain sensor and retrieve its values is mandatory. The Sensor Access Service is strongly related to the OGC Sensor Observation Service and therefore provides similar functionality. The Sensor Access Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *SensorAdministration:* Allows the client to add or remove sensors at the ser-

	vice and also change the descriptions of already existing sensors. • *SensorConfiguration:* Provides functionality that allows the client to configure a specified sensor (e.g.: adjust measurement range, position) • *SensorData:* Allows the client to query for sensors that provide a specific functionality/type of measurement and retrieve these measurements.
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about the common and specific capabilities of a Sensor Access OSI. Examples of the specific capabilities are: • *configurationSupported*: Flag whether the *SensorConfigurationInterface* is implemented • *administrationSupported*: Flag whether the *SensorAdministrationInterface* is implemented • *configurationCacheSupported*: Flag whether the checkSensorConfiguration operation caches valid configurations. • *cacheTimeout*: Defines the duration of time after which a cached configuration will be deleted and the associated *OA_SensorConfigurationID* is invalid
Interface *SensorAdministration*	
addSensor	Add a new sensor with its specified description to the services.
updateSensorDescription	This operation can be used to change the description of an already existing sensor.
removeSensor	Removes the specified sensor from the service.
setSensor Data	Publishes new sensor data at the service so that clients may retrieve it through an invocation of the *getSensorData* operation.
Interface *SensorConfiguration*	
getConfiguratioSchema	Retrieves the configuration schema of the specified sensor. The schema describes format, mandatory and optional parts of a valid sensor configuration.
getSensor Configuration	Retrieves the currently active configuration for the specified sensor.
setSensor Configuration	Sets the configuration for the specified sensor.
Interface *SensorData*	
getSensor	Retrieves a list of identifiers of those sensors that match the specified requirements. These requirements are formulated in a query language. The query language is indicated in the service's capabilities.
getSensor Data	Retrieves actual data (real measured or calculated/simulated data) of the specified sensor.
getSensorDataTypes	This operation returns the schemas for the data types that can be retrieved at this service.
Example usage	A sensor administrator wants to publish ozone measurement values so that an environmental authority can retrieve it and produce a report.
Comments	The Sensor Access Service is a very basic service that does not include planning of series of measurements or notifications. Notifications can be supported by implementing the *notify* operation of the *AsynchronousInteraction* interface of the OA Basic Service on the client side (see section 9.6.1).

Table 18: Description of the Sensor Access Service

9.7.5 Catalogue Service

Name	Catalogue Service
Standard Specifications	The ORCHESTRA Catalogue Service has been derived from the approach how meta-information is being handled in the OA (see section 8.4). Thus, the following series of catalogue standards has been considered, but the goal has not been to specify a service that is exactly compliant to one of these services. However, the functionality of the following standards for basic search and publication is supported by the ORCHESTRA Catalogue service such that it may be mapped on corresponding service implementations. In addition, the ORCHESTRA Catalogue Service additionally provides a navigation interface for navigation in the catalogue contennt • OASIS UDDI Version 3.0.2 Specification (http://uddi.org/pubs/uddi_v3.htm) • OGC 04-021-r3 Catalogue Service Implementation Specification V2.0.1 (Class: Abstract Specification) • OGC 04-017r1 Catalogue Services – ebRIM (ISO/TS 15000-3) profile of CSW (CAT2 AP ebRIM) V0.9.1 (Class: Engineering Specification) • OGC 04-038r2 ISO19115/ISO19119 Application Profile for CSW 2.0 ((CAT2 AP ISO19115/19)) V0.9.3 (Status: Best Practices) • OGC 06-079r2 EO Application Profile for CSW 2.0 (Status: Pending) • OGC 06-131 EO Extension Package for ebRIM (ISO/TS 15000-3) Profile of CSW 2.0 (Status: Discussion Paper) The ORCHESTRA Catalogue Service does not define a meta-information schema by itself. The intention of the ORCHESTRA Catalogue is to provide a flexible service type which can be adapted to the particular purposes of the application environment.
Description	The Catalogue Service supports the ability to publish, query and retrieve descriptive information (meta-information) for resources (i.e. data and services), meta-information about ORCHESTRA Source Systems (just like meta-information for other ORCHESTRA services) and instances of feature types that are referred to by extensions of the OMM_FeatureType, such as documents, schemas, dictionaries, equations and models. The Catalogue Service is not tied to a particular schema of a meta-information standard (e.g. ISO 19115); instead it supports application schemas for meta-information (OAS-MI) that are designed according to the rules of the OMM. Due to independence from a specific meta-information standard the catalogue can be used to store meta-information about services and data according to the meta-information schema used in the catalogue. Therefore a catalogue instance can be used as a data catalogue, service registry or both if multiple meta-information types are used in the catalogue instance. The multilinguality of the catalogue is dependent on the multilingual capabilities of the meta-information schema used inside the catalogue. Meta-information entries in catalogues represent resource characteristics that can be queried and presented for evaluation and further processing by both humans and software. The Catalogue Service supports the discovery of registered resources within an information community and returns binding information that allows a user to locate and access the resource (e.g. an URI). The Catalogue Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *CatalogueSearchInterface:* The interface for search provides a means for searching information in the catalogue. The client asks the catalogue capabilities for the available catalogue entry types. Each entry type is associated

	with a meta-information type and its corresponding query languages. With this information the client can query the catalogue entry type with the appropriate query language. • *CataloguePublicationInterface:* The interface for publication is responsible for including, updating and deleting meta-information in the catalogue. It is pushing information into the catalogue. It provides operations for filling the catalogue. The needed meta-information could be created with some kind of meta-information editor, in which the user is specifying the meta-information about resources to be registered in the catalogue, or it could be collected through the collection interface. • *CatalogueCollectionInterface:* The collection interface provides operations, which are helpful for the automatic update of catalogue content in difference to the publication interface, which just fills the catalogue with given content. It is pulling meta-information into the catalogue. The operations in this interface should be able to be triggered from the outside of the catalogue and it should be possible to define a periodic update from the catalogue content. • *CatalogueNavigationInterface:* With the means of this interface, the user is looking for meta-information records managed by the catalogue by navigating from node to node. The search is driven by the catalogue itself: no query is performed. Note that the implementation of this interface makes the Catalogue Service a stateful service. • *AsynchronousInteraction (OA Basic Service):* Definition of a uniform way to request asynchronous execution of a service operation, e.g., for operations which are time-consuming or deliver results periodically. This interface is used by the *collectMetaInformationPeriodic* operation of the *CatalogueCollectionInterface*.
Interface *ServiceCapabilities*	
getCapabilities	Informs the requestor about the common and specific capabilities of a Catalogue Service instance. Examples of specific capabilities are the information about query languages, the statement if the catalogue service instance is the main catalogue of an OSN (the “OSN Catalogue” as introduced in section 11.1.3) and the meta-information types used in the Catalogue Service instance.
Interface CatalogueSearchInterface	
search	Returns a list of identifiers for corresponding features, given a request expressed in a given query language.
getMeta Information	Returns associated meta-information instances, given some identifiers of features managed by the catalogue as returned by a previous search operation call.
getQuery Domain	Returns the domain of values that are applicable to a property of the meta-information type. This is used by catalogue clients. Using this operation by giving the parameters of interest, the client shall know what values (e.g. list of values, range of values) are allowed for a meta-information property.
getMeta Information Type	Returns the associated meta-information type, given a list of catalogue entry types managed by the catalogue.
Interface *CataloguePublicationInterface*	
createMeta Information	Pushes information into the catalogue. The task of this operation is to insert catalogue content into the catalogue. The operation receives the meta-information to be stored and returns information about the update of the catalogue.
setMeta Information	Updates the catalogue content. The operation receives the meta-information types to be stored and returns information about the update of the catalogue.
deleteMeta	Deletes catalogue content from the catalogue. The input is a constraint to identify

Information	the catalogue content, which needs to be deleted. The operation returns information about the update of the catalogue.
Interface *CatalogueCollectionInterface*	
collectMeta Information	Pulls meta-information into the catalogue. The operation receives one reference of a source of meta-information and a catalogue entry type. This catalogue entry type is the type in which the meta-information is going to be stored in the catalogue. The operation returns information about the update of the catalogue.
collectMeta Information Periodic (optional)	Receives one reference of a source of meta-information, the catalogue entry type and the time interval between two collections and a date to stop the collect. The catalogue entry type is the type in which the meta-information is going to be stored into the catalogue. The operation is processed periodically according to the given intervals and stores the resulting meta-information into the catalogue. The operation should be called asynchronously using the *AsynchronousInteraction* interface. The operation returns information about the update of the catalogue.
Interface *CatalogueNavigationInterface*	
getNavigationRoots	Returns the catalogue entries that can be used to start navigation inside the catalogue. If none is returned, no navigation will be possible.
getNavigationEdges	Returns all relationships that start from this node to other ones given an existing node in the catalogue. Each relationship is annotated by the kind of relationship, which adds some semantic information (e.g. broader, narrower, similar) to the link.
Interface *AsynchronousInteraction*	
invokeAsync	Starts asynchronous execution of the *collectMetaInformationPeriodic* operation of the *CatalogueCollectionInterface*. The *invokeAsync* operation returns immediately with an identifier (invocation ID) representing the asynchronous execution.
abort	Aborts execution of the previously invoked asynchronous *collectMetaInformationPeriodic* operation identified by its invocation ID.
notify	Passes a notification to the callback interface provider of the *CatalogueCollectionInterface*.
Example usage	A possible usage scenario of the catalogue is the usage of a catalogue for discovering maps and displaying them in a map viewer. The following steps need to be accomplished for this scenario: 1. The catalogue needs to be initialized with meta-information about the maps and a service capable of displaying the maps. The meta-information can be written into the catalogue using operation *createMetaInformation*. 2. The user performes a search for available maps on the catalogue using the *search* and *getMetaInformation* operations. 3. The user performes a search for an available map viewer, again using the *search* and *getMetaInformation* operations. 4. The user displays the maps in the map viewer, using the retrieved meta-information about the maps and the map viewer.
Comments	The abstract specification leaves the question of the meta-information creation open. It could be created by the user with the help of a meta-information editor or automatically either within the catalogue inside *collectMetaInformation* or with the usage of other means and services inside *collectMetaInformation*. The support of multi-linguality depends on the meta-information schema used in the catalogue. Meta-Information about data and services inside the scope of an OSN will be described with the help of the service capabilities.

Table 19: Description of the Catalogue Service

9.7.6 Name Service

Name	Name Service
Standard Specifications	• IETF RFC 1034 Domain Names - Concepts and Facilities • IETF RFC 1035 Domain Names - Implementation and Specification
Description	The objective of the Name Service is to encapsulate the implemented naming policy for service instances in an OSN. It is responsible for creating globally unique OSI names using a defined naming policy, e.g. by mapping between OSI names and corresponding platform-specific service identifiers. If the naming policy requires additional information to ensure uniqueness of names, e.g. an OSN name, then such information may be provided by configuration and shall be hidden at the service interface. A central Name Service instance for all OSNs is not required. Instead, there may be multiple Name Service instances, and each one may use a different naming policy, as long as global uniqueness of created names is guaranteed. If multiple Name Service instances are available within an OSN, they shall be related, i.e. each one can be used for name resolving within the OSN. It is possible to share a Name Service instance among multiple OSNs. Within an OSN that is based on multiple service platforms, a Name Service instance is available for each service platform and shall be used for name resolving within that platform. The Name Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *NameCreationAndResolution*: provides operations to create names and to resolve names given a platform-specific identifier (PSI) or vice-versa. • *NamingServiceLinkage*: provides operations to support the linkage between several Name Service instances.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the common and specific capabilities of a Name Service instance. An example of a specific capability is the naming policy that is applied in the Name Service instance.
Interface *NameCreationAndResolution*	
registerService	An OSI is made known to the Name Service. The OSI is specified by its platform-specific service identifier (PSI). It is related to the current service platform, i.e. the platform on which the Name Service is based. The operation returns a globally unique name for the OSI according to the implemented naming policy. From that point on, name resolution is possible for that OSI name and PSI. If a PSI is not provided as input parameter, an OSI is registered which has not yet an assigned PSI. In that case, it is assumed that the Name Service itself assigns a PSI to the OSI This PSI can be retrieved later by means of the *getPSI* operation.
getPSI	Given an OSI name, the PSI of that OSI is returned if known to the Name Service. The PSI is used to access the OSI within the current service platform. It may therefore be a PSI of a service gateway, if the OSI is based on a different platform.
getName	Given the PSI of an OSI, the name of that OSI is returned if known to the Name Service.

Interface *NamingServiceLinkage*	
linkName Service	This operation establishes a linkage between this Name Service instance and another one which is specified by its PSI within the current service platform. The linkage is used to allow for cascading name resolving. This means if this Name Service instance has no information to map an OSI name to a PSI, or vice versa, it can redirect the request to all linked Name Service instances.
unlinkName Service	This operation removes a linkage between this Name Service instance and another one which is specified by its PSI within the current service platform.
Example usage	An instance of a Name Service is useful in the case of OSNs that span multiple service platforms connected through an OSN gateway.
Comments	none

Table 20: Description of the Name Service

9.7.7 User Management Service

Name	User Management Service
Standard Specifications	• IETF RFC 2251 Lightweight Directory Access Protocol (LDAP) (v3) The LDAP RFC standards span from RFC 2251 to RFC 2256. The following RFC has been used as a template to define subject attributes in the ORCHESTRA User Management Service: • IETF RFC 2256 - A Summary of the X.500(96) User Schema for use with LDAPv3
Description	The User Management Service is used to create and maintain subjects including groups (of principals) as a special kind of subjects. In general, subjects represent entities that need to be authenticated. They are not authenticated themselves but rather represent a point of contact and management feature for authentication and authorisation purposes. A subject is decoupled from authentication. This decoupling is done by separating principals from subjects. A principal is an identity of a subject and is defined in an Authentication Service instance. Management of subjects includes the association to principals as well as storage of subject attributes. Group management includes definition of principal memberships. The User Management Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *UserManagementService*: Management of subjects and group subjects.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the common and specific capabilities of a User Management Service instance. Examples of specific capabilities are structural information on subject attributes specialised with respect to the different types of subjects: • for human users, e.g. first name, surname as well as contact information • for groups, e.g. administrative contact. • for services, e.g. administrative contact. Additional pieces of information may be defined by a policy provided by the respective OSN.
Interface *UserManagementService*	

createSubject	Creates a subject. After a subject has been created, at least one principal has to be created and associated with the subject.
deleteSubject	Deletes a subject including the deletion of all associated principals and subject attributes.
updateSubject	Updates the subject itself. Can be used to change subject related information, e.g. subject attributes.
createGroup	Creates a group. Groups contain principals, not subjects. After creation a group has no members. Since a group is a special subject, principals have to be added. These can be managed using the *addPrincipalToSubject* and *removePrincipalFromSubject* operations. Group principals represent the identities of the group not group members. Group members can be managed using the operations *addPrincipalToGroup* and *removePrincipalFromGroup*.
deleteGroup	Deletes a group without deleting group member principals. Principals of the group are deleted if not specified otherwise.
updateGroup	Updates the group. Can be used to change group related information, e.g. group attributes. In order to manage group memberships use the operations *addPrincipalToGroup* and *removePrincipalFromGroup*.
getGroups	Retrieves an enumeration of existing groups.
addPrincipalToSubject	Associates an existing principal to an existing subject. This operation can also be used for the assignment of principals to group subjects (not group members).
removePrincipalFromSubject	Removes a prior assigned principal from a subject. This operation can also be used to remove principals from group subjects (not group members).
getSubjects	Enumerates all subjects of the current service instance. Use the operation getGroups to exclusively retrieve group subjects. There is no operation to retrieve an enumeration of non-group subjects. This can be done by simply removing group subjects from the result.
removePrincipalFromGroup	Removes the association between a given principal and a given group. The removed principal is not deleted in the corresponding Authentication Service.
addPrincipalToGroup	Associates an existing group with an existing principal. The principal may reside in another User Management Service instance.
Example usage	A group of users concerned with forest fires manages maps describing fire damage. Another group of users working on flood risk analysis would like to access the maps because they are relevant for their planning. Therefore, read access is granted to the flood analysis group for all maps and features contained in the map layers managed by the forest fire group.
Comments	none

Table 21: Description of the User Management Service

9.7.8 Authorisation Service

Name	Authorisation Service
Standard Specifications	The following standard describes the main ideas of role based authorisation systems: • Ferraiolo David F. et. al: Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security, Vol. 4, No. 3, Au-

	gust 2001, Pages 224–274. http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf The Authorisation Service implements the ideas of what is called “Core Role-based Access Control (RBAC)” in the NIST standard as close as possible. A further source of inspiration has been the following RFC as it has many requirements in common with ORCHESTRA UAA: • IETF RFC 2704 The KeyNote Trust-Management System Version 2 (September 1999) http://www.ietf.org/rfc/rfc2704.txt?number=270
Description	The Authorisation Service gives a compliance value as response to a service requesting an authorisation decision for a given authorisation context. The Authorisation Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *AuthorisationService*: Includes all operations which are common to all Authorisation Service implementations regardless to their underlying paradigms. • *XAuthorisationAdministration* (where *X* could be e.g. *Rbac* or *Principal*): The administration interface is specific to the underlying paradigm, e.g. supporting role management and thus may vary for different Authorisation Service implementations. In the following a representative administration interface for a role based Authorisation Service is presented.
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about the common and specific capabilities of an Authorisation Service instance. Examples of specific capabilities are the supported authorisation paradigms (e.g. principal permissions, or role-based access control). These paradigms are accompanied by specialised by dedicated administrative interfaces.
Interface *AuthorisationService*	
authorise	Requests an authorisation decision for a given authorisation context. An authorisation context is required as an input parameter. An authorisation context is a set of information used by the authorisation service to determine the authorisation decision for a given request. The authorisation context can contain, for example, the requesting principal(s), name of the invoked operation, etc. A compliance value representing the advice how to treat a certain service request is delivered as an output parameter. Authorisation contexts and compliance values need to be agreed upon between a service and its Authorisation Service.
Interface *Administration*	
createRole	Creates a new role. Newly created roles are empty. Neither permission nor principals are assigned, yet.
deleteRole	Deletes an existing role. Permission and principal assignments are deleted as well.
getRoles	Retrieves an enumeration of existing roles.
updateRole	Updates an existing role, e.g. description, etc.
assignPermissionToRole	Assigns permission to a certain role. Permission and role have to exist already.
unassignPermissionFromRole	Removes permission from a certain role.
assignRoleToPrincipal	Assigns an existing role to an existing principal. This indirectly assigns permissions associated with the role to the principal.

unassignRole FromPrincipal	Removes the given role from a certain principal. This indirectly removes permissions associated with the role from the principal.
Example usage	For a Format Conversion Service it may be necessary to restrict access to certain principals. The service provider might use an Authorisation Service to assign these principals' permissions to perform conversions. This could be done with a service type independent Authorisation Service implementation supporting operation level authorisation. The authorisation context of such a service needs to include at least requesting principal(s) as well as the requested operation. An Authorisation Service implementation which is specific to Format Conversion Services might additionally restrict the size of files to be converted depending on the requesting principal. The authorisation context for such a scenario would need to include the size of the file to be processed. In the domain of Risk and Crisis Management, another example is the following: Access rights like read, write, access, execute services, compose services or feature collections, modify rights etc. are granted to principals of a Civil Protection Agency for all resources that relate to the responsibility domain of the agency. In case of a hazard event, read access rights are extended to all resources related to the hazard, independent of their organisational assignment.
Comments	none

Table 22: Description of the Authorisation Service

9.7.9 Authentication Service

Name	Authentication Service
Standard Specifications	The following RFC standards have been taken into consideration as individual authentication mechanisms. The abstract ORCHESTRA Authentication Service Specification is intended to be independent from authentication mechanisms. Its current implementation uses a non-encrypted username/password mechanism but could also integrate a Kerberos authentication mechanism as described in RFC 4120. • IETF RFC 4120 - The Kerberos Network Authentication Service (V5) • IETF RFC 4158: Internet X.509 Public Key Infrastructure: Certification Path Building • IETF RFC 4210: Internet X.509 Public Key Infrastructure Certificate Management Protocols • IETF RFC 4211: Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF) • IETF RFC 4325: Internet X.509 Public Key Infrastructure Authority Information Access Certificate Revocation List (CRL) Extension • IETF RFC 4386: Internet X.509 Public Key Infrastructure Repository Locator Service • IETF RFC 4387: Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP
Description	The Authentication Service verifies genuineness of principals using a set of given credentials. The authentication mechanism, which means the way authentication is performed, is up to the service implementation. Which credentials an Authentication Service needs as well as the way they are passed is specific to the authentication mechanism used. Session information returned after a successful authentication can be used to invoke services demanding authenticated principals. A service might use this informa-

	tion to perform authorisation requests. The Authentication Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *AuthenticationService*: Includes all operations which are common to all authentication mechanisms. • *UsernamePasswordMechanism*: Contains operations which are specific to the authentication based on a username/password authentication mechanism. This interface should specify credentials as well as the way they are passed.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the common and specific capabilities of an Authentication Service instance. Examples of specific capabilities are the supported authentication-mechanisms (e.g. username-password authentication, public-key authentication).
Interface *AuthenticationService*	
login	Initiates the validation of a certain principal for given credentials. Credentials have to be passed using the *AuthenticationMechanism* interface before calling the login operation. This needs to be done within a transaction. As an output parameter, the session information that can be used to invoke services demanding authenticated principals is provided.
addPrincipal	Creates a new principal. The principal representation is specific to the authentication mechanism used. For a username/password authentication the principal contains at least a username.
remove Principal	Deletes an existing principal. Removal of principals should not be done without updating corresponding User Management OSIs (see section 9.7.7) as well as updating services having permissions associated to the principal to be deleted. A solution to this could be the use of administration tools to keep track of consistency.
update Principal	Updates an existing principal. The principal to be updated as well as information to be changed, e.g. new username, shall be provided as input.
add Credentials	Adds credentials to a certain principals. Credentials are specific to the authentication mechanism used. For a username/password authentication credentials is a password.
Update Credentials	Updates credentials for a certain principal. The principal (username) for whom the credentials (password) should be changed as well as changed credentials shall be provided as input.
deactivate Principal	Deactivates a principal without removing it. The principal, e.g. username to be deactivated and additional information, e.g. a time period for deactivation, shall be provided as input.
activate Principal	Activates an existing principal. The principal, e.g. username to be activated and additional information, e.g. a point of time for activation, shall be provided as input.
Interface *UsernamePasswordMechanism*	
setUsername	Used to pass the principal to be authenticated. In a username/password authentication the username represents the principal.
setPassword	Used to pass the credentials to verify authenticity. In a username/password authentication the password represents credentials.

Example usage	A Format Conversion Service demands authorisation based on principals. Therefore each service requestor has to pass session information including at least one authenticated principal. In order to invoke a service a subject needs to authenticate a principal having appropriate permissions. The resulting session information can be passed to the service. The service uses – among others - the session information to build the authorisation context which is passed to the Authorisation Service.
Comments	It is part of the characteritsics of an OSN to determine if user authentication is necessary and if so, by using which authentication mechanism.

Table 23: Description of the Authentication Service

9.7.10 Service Monitoring Service

Name	Service Monitoring Service
Standard Specifications	• Web Notification Service 03-008r2
Description	The Service Monitoring Service provides an overview about ORCHESTRA Service Instances (OSIs) currently running within an OSN. OSIs can either be monitored using a push or pull model, that is, the status information is actively retrieved from an OSI by a service (this could be any service but preferably the Service Monitoring Service) or they are sent to the Service Monitoring Service. There is also the possibility to register an alert service and bind information of a specific monitoring status to that alert service. That way every time such information is received the alert operation of the alert service will be invoked. The Service Monitoring Service provides the functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *ServiceMonitoringService*: Implements a push model monitoring and alert service binding • *Monitorable Interface:* A service must implement this interface in order to use the pull model monitoring. • *Alert Interface:* Used when monitoring values of a certain status are provided. This can for example be used to contact the service administrator via email or Short Message Service.
Interface *ServiceCapabilities*	
getCapabilities	Informs the client about the common and specific capabilities of a Service Monitoring Service instance. Examples for specific capabilities are the supported statistics about the usage of a service in an OSN.
Interface *ServiceMonitoringService*	
putStatus	Gives any service the possibility to send monitoring information to the monitoring service.
getConfiguration	Retrieves the current configuration of the monitoring service.
setConfiguration	Sets the current configuration of the monitoring service. This includes information such as which services should be monitored, the binding between status information and alert services.

getConfigurationSchema	Retrieves the schema that describes the format of the configuration.
getStatistics	Retrieves statistical information about the monitored OSN or single services. These statistical values are features in order to enable easy usage with other feature processing services.
Interface *Monitorable*	
getStatus	Retrieves the status of a specific monitored property of the implementing service.
getConfiguration	Retrieves the currently active configuration of the monitored service.
setConfiguration	Sets the current configuration of the monitored service (e.g., interval that must be between getStatus calls in order to have new values available)
getConfigurationScheme	Retrieves the schema that describes the format of the configuration.
Interface *Alert*	
alert	This operation does not have a predefined functionality. It can either be sending an email or a Short Message Service or do some other mandatory processing.
Example usage	A service provider has her FeatureAccessService monitored by the ServiceMonitoringService. Whenever the hard disk usage exceeds 90% of the storage available a monitoring value of status CRITICAL is produced. This value is retrieved by the ServiceMonitoringService and since the status has been bound to an alert service, it is sent there invoking the alert operation. This OSI that implements the Alert Interface then sends a ShortMessageService to the service operator who can react to this situation.
Comments	Since the concrete procedure of reaction to an alert is application and most likely company dependant the semantic meaning of the alert operation can't be given. In some cases a simple email or other message will be passed to a responsible person, in other cases some complex automatic reaction will take place in case of an alert.

Table 24: Description of the Service Monitoring Service

9.8 OA Support Service Descriptions

9.8.1 Coordinate Operation Service

Name	Coordinate Operation Service
Standard Specifications	• ISO 19107:2003 Geographic information -- Spatial schema • ISO 19111:2003 Geographic information -- Spatial referencing by coordinates • OGC 05-008c1 Web Services Common Specification V1.0 • OGC 05-013 Web Coordinate Transformation Service (WCTS) draft Implementation Specification (Discussion Paper)
Description	The Coordinate Operation Service changes coordinates on features from one coordinate reference system to another (based on a 1-1 relationship). This includes operations on datum and projection. A Datum is used as a basis for defining a coordinate reference system and it specifies how the coordinate system is related to the earth. Examples are WGS84 and NAD1950. A projection is a method for depicting 3-dimensional data (the shape of the earth) in 2 dimensions.

	There are two principal variants of coordinate operations: • Coordinate conversion: An operation on coordinates that does not include any change of Datum. Examples of a coordinate conversion are a map projection between projected coordinates and geographic coordinates, or change of units such as from radians to degrees or feet to meters. • Coordinate transformation. An operation on coordinates that usually includes a change of Datum. The parameters of a coordinate transformation are empirically derived from data containing the coordinates of a series of points in both coordinate reference systems. This operation introduces errors, hence allowing derivation of error (or accuracy) estimates for the transformation. The Coordinate Operation Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *CoordinateOperation:* Request to change coordinates of features, either by a coordinate conversion or a coordinate transformation.
Interface *ServiceCapabilities from (OA Basic Service)*	
get Capabilities	Informs the client about the common and specific capabilities of Coordinate Operation Service instance. Examples of specific capabilities are the supported conversions and transformations.
Interface *CoordinateOperation*	
check Operation	Reports if an operation between two Coordinate Reference Systems is supported by the service implementation and, if so, if it is a conversion or a transformation.
convert Coordinates	Convert coordinates without any change of Datum.
transform Coordinates	Transform coordinates usually including a change of Datum.
Example usage	Coordinate conversion: A user wants to convert coordinates from UTM Zone 33, Euref89 to Geographic coordinates, Euref89. Coordinate transformation: A user wants to change coordinates from UTM Zone 33, ED50 to Geographic coordinates, Euref89
Comments	none

Table 25: Description of the Coordinate Operation Service

9.8.2 Gazetteer Service

Name	Gazetteer Service
Standard Specifications	• ISO 19111:2003 Geographic information -- Spatial referencing by coordinates • ISO 19112:2003 Geographic information -- Spatial referencing by geographic identifiers • OGC 05-035r2 Gazetteer Service - Application Profile of the Web Feature Service Implementation Specification V0.9.3 (Best Practices Paper)
Description	The Gazetteer Service allows a user to relate a geographic location instance fied by geographic names (e.g. city, lake, region, street) with an instance identified by coordinates (e.g. a point, line, polygon or sets of these). A client delivers geographic names or describes them indirectly by means of a query (e.g. all cities in

	Bavaria) and receives geographic objects with their corresponding coordinates or vice versa. The Gazetteer Service usually provides this functionality by accessing a directory of geographic identifiers that describes location instances, called a gazetteer. The conceptual model of the gazetteer is taken from ISO 19112:2003. Here, location instances contain both geographic identifiers and the geographic positions. Access to the gazetteer is performed through operations of the *vice* interface (see section 9.7.1). Thus, by the selection of location instances using the query mechanisms of the Feature Access Service the relationship between names (indirect spatial reference) and coordinates (direct spatial reference) is ried out. For the purpose of gazetteer maintenance, the Gazetteer Service supports changes and updates of a gazetteer, too. A sequence of these operations may, if required, be secured by a transactional interface. The Gazetteer Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *FeatureAccessService*: provides read and write access to a gazetteer. • *TransactionInterface:* Secures sequences of change requests to a gazetteer.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the capabilities of a Gazetteer Sesrvice instance. Examples of specific capabilities are the provider organisation, the version and the geographic scope of the gazetteer.
Interface *FeatureAccessService (from Feature Access Service)*	
	The operations of the *FeatureAccessService* interface are used to access to the location types and instances of a gazetteer.
Interface *TransactionInterface*	
	The operations of the *TransactionInterface* are used when a synchronised access to the gazetteer must be assured, especially in the case of the *setFeature, createFeature and deleteFeature* operations.
Example usage	The Gazetteer Service may be used to integrate information in a risk assessment process if one of the source information items is geo-referenced by a geographic identifier (e.g. a statistical result based on a departmental area) and another by a geographic coordinate (e.g. measurement values at monitoring locations). In this scenario, the Gazetteer Service helps to generate comparable information that may be commonly processed.
Comments	A future version may consider a combination of a gazetteer with a thesaurus. Thus, the Gazetteer Service may use the operations of the Thesaurus Access Service (see section 9.8.6) in order to support multi-lingual gazetteers and fuzzy queries based on synonyms, quasi-synonyms or related terms, like “give me the coordinates of the city by the riverside of the Rhine that is close to Wiesbaden”. Further enhancements may cover distributed gazetteers, possibly across borders i.e. in combination with the gazetteer-thesaurus combination discussed above.

Table 26: Description of the Gazetteer Service

9.8.3 Annotation Service

Name	Annotation Service
Standard Specifications	• W3C OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/

	• W3C-Resource Description Language http://www.w3.org/RDF/ • W3C RDF-Schema http://www.w3.org/TR/rdf-schema/
Description	The Annotation Service automatically generates specific meta-information from various sources and relates it to semantic descriptions. Semantic descriptions are to be specified as elements of an ontology (e.g. concepts, properties, instances). Sources to be annotated can contain unstructured information (e.g. documents, texts) or structured information (e.g. databases, applications). Annotations refer to the concepts of an ontology, which is specified in an ontology language such as OWL and RDF-Schema (a subset of OWL). The content of an annotation can be stored as a simple string. In order to provide references to concepts, instances and relation types stored in either a knowledge repository or a data ontology, the RDF syntax can be used. The generation of annotations of *unstructured sources* is based on automatic Information Extraction, by means of which named entities occurring in documents and texts can be identified and normalized by means of Natural Language Processing. The process of extracting information and its assignment to ontological elements is based on background knowledge held in a repository, the (pre-populated) knowledge base. In an OSN, such a knowledge base is accessible by means of the Knowledge Base Interface (see section 9.6.5). In addition to named entity identification, the service can automatically discover and formalize new knowledge by analyzing the texts. In a certain application scenario, this knowledge can be used to populate a knowledge base, from where it can be queried by means of query languages. The semantic annotation of documents and texts enables applications such as highlighting and document viewing. The Annotation Service can automatically generate meta-information for *structured sources* such as databases, applications, etc. As a pre-requisite of the annotation service, the structure and content of such a resource is to be transformed into a data ontology which is compliant with the ontology containing the semantic descriptions. An annotation is a mapping of an element of this ontology to an element of the data ontology. The semantic annotation of databases and applications enables applications such as exploration of the database structure and content by means of ontology query languages, or interpretation of query results by means of domain knowledge. The Annotation Service provides its functionality through the following interfaces: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *AnnotationService*: For sources to be annotated (e.g. documents, databases), an additional document - called a "semantic document" - is established which contains the annotations. Another operation of the service allows annotation of texts; here, the annotations are delivered directly in the operation result; a semantic document is not generated.
Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the common and specific capabilities of an Annotation Service instance. Examples of specific capabilities are the supported annotation strategies (identification, population of new knowledge etc.), a list of mime types of documents which can be annotated, and a list of supported data and domain ontology formats.
Interface *AnnotationService*	

create Semantic Document	In a first step prior to annotation, a "semantic document" is associated with the base document. A semantic document contains the content of the base document, the annotations and links to the base document and the corresponding domain ontology. After creation, a semantic document only contains the content of the base document; the annotations and the links are entered through the *annotateDocument* resp. *annotateDataOntology* operations.
annotate Document	Generates annotations for a given semantic document for an unstructured source. The generated annotations are inserted into the semantic document.
annotateText	Generates annotations for a given text "on the fly", i.e. they are not stored in a semantic document.
annotateData Ontology	Generates annotations for a given semantic document for a structured source. The semantic document has previously been generated from its data ontology by means of a *createSemanticDocument* operation. The generated annotations are inserted into the semantic document.
Example usage	Risk maps usually can display various thematic layers. The graphical representation in the risk map is explained in an attached legend. In many cases, the user needs more textual explanation about what the values in a legend exactly mean. With a growing number of layers and legends, a map can contain a considerable amount of attached text; new layers, legends and texts can be added dynamically. Moreover, the text itself could contain technical terms that make it difficult to read, or users might only be interested in getting further information on items occurring in the text. In this scenario, the attached text could be processed in an *annotateText* operation, which automatically sets up links of the terms occurring in the text to elements (concepts, instances) described in a domain ontology. The user can navigate to the respective ontology element and start browsing the ontology, thus getting help for interpretation of the text.
Comments	The service does not maintain the set of sources that are to be annotated; this functionality is expected to be provided elsewhere. For instance, annotation could be performed on a regular basis by means of a background job triggered at times of low load. The job checks the set of sources for changes that have been performed since the last run. Documents which have been changed are annotated again and old annotations are deleted.

Table 27: Description of the Annotation Service

9.8.4 Format Conversion Service

Name	Format Conversion Service
Standard Specifications	The following language is used to select data formats: • MIME Media Types (http://www.iana.org/assignments/media-types/)
Description	The Format Conversion Service allows the conversion of data given in one format to the corresponding data given in another format. Each conversion between a pair of formats requires a conversion algorithm. The problem is how two organisations are able to exchange their data (e.g. documents) without caring about the format the other side uses. This is the reason why the Format Conversion Service is needed. It allows the conversion from one data format (in case of documents e.g. MS-Word, OpenDocument, pdf,) to another one in order to easily exchange data between different organisations. Data could be text based, like a word document or a pdf, or it could be binary data like JPEG or WMF. The Format Conversion Service provides its functionality through the following interfaces:

	• *ServiceCapabilities*: Informs about the common and specific capabilities. • *FormatConversion*: Provides the conversion operations.
Interface *ServiceCapabilities*	
get Capabilites	Informs the client about the common and specific capabilities of a Format Conversion Service instance. Examples of specific capabilities are the supported source and target formats and the conversion functionality between these formats.
Interface *FormatConversion*	
convert	Performs the conversion given by input and output MIME type.
Example usage	A time series of measurement values is available as an MS-Excel sheet and shall be converted into an XML file for further processing in an RM application.
Comments	It will be possible to build chains of format conversions. Example: If the conversion functionality png2gif, gif2jp and jpg2pdf are available, the call *convert*(doc1, png, pdf) will directly convert form a png to a pdf format.

Table 28: Description of the Format Conversion Service

9.8.5 Schema Mapping Service

Name	Schema Mapping Service
Standard Specifications	No standard service specification currently exists, on which the functionality of the Schema Mapping Service could be based. Several standards exist, on which a language for describing a schema mapping can be based. However, as the Schema Mapping Service does not define a specific schema mapping language, it is up to the implementation specification to define these. Prominent (draft) standards which can be used for describing a schema mapping are: • W3C XSL Transformations (XSLT), version 1.0 (http://www.w3.org/TR/xslt/) • XQuery 1.0: An XML Query Language, W3C Recommendation (http://www.w3.org/XML/Query/) • W3C SPARQL Query Language for RDF, W3C Working Draft, 4 Oct 2006 (http://www.w3.org/TR/rdf-sparql-query/)
Description	The Schema Mapping Service provides functionality that is related to the mapping of features from a source into a target schema. It provides this functionality through two interfaces. The main functionality of the *SchemaMapping* interface is to execute a schema mapping. A schema mapping is considered to be “the definition of an automated transformation of each instance of a data structure A into an instance of a data structure B that preserves the intended meaning of the original information”. The service takes a feature collection and a description of the mapping from the source to the target schema as input and returns the features in the target schema. A schema mapping is described by • an identifier that is unique to the Schema Mapping Service instance; • descriptions of the source and target feature types; • the schema mapping language used to describe the mapping; and • a reference to the actual mapping.

	The Schema Mapping Service can be used to (1) directly map from one application schema to another one, or (2) to map from an application schema to a common (or community) schema (or vice versa). The latter can be used to perform an indirect mapping between two application schemas through the community schema. The mapping of features might also require that several feature collections be combined. In order to support this, an optional concatenation operation is also included in the interface. The description of the schema mapping is required as an input. It is outside the scope of the Schema Mapping Service to automatically derive a mapping between two application schemas. The *SchemaMappingRepository* interface supports repository functionality for mappings between source and target feature types. Service can also serve as a repository for mappings between source and target feature types. For this, operations for the creation (registration), retrieval, updating and deletion of schema mapping descriptions are foreseen. The Schema Mapping Service provides its functionality through the following interfaces: • *ServiceCapabilities:* Informs about the common and specific capabilities. • *SchemaMapping:* Execution of schema mappings and concatenation of feature collections. • *SchemaMappingRepository*: Creation, deletion, update and selection of schema mappings.
Interface *ServiceCapabilities*	
getCapabilities	Informs the requestor about the common and specific capabilities of a Schema Mapping Service instance. Examples of specific capabilities are the supported schema mapping language (for the *Schema Mapping* interface) and a list of the mappings registered with the service (for the *Schema Mapping Repository* interface).
Interface *SchemaMapping*	
mapFeatures	Maps a feature collection to a target schema.
concat	Concatenates several feature collections.
Interface *SchemaMappingRepository*	
createMapping	Registers a new mapping with this instance of the Schema Mapping Service.
getMapping	Returns a (list of) mapping(s) matching a given query.
setMapping	Updates a specific mapping.
deleteMapping	Deletes all mapping matching a given query.
Example usage	A client wants to transform a data source in a local schema into a common agreed global schema. The client submits a feature collection and mapping rules specifying how to map the features into the required feature type.

Comments	The described interfaces can be used in service implementations in different ways: • A service that only implements the *SchemaMapping* interface can be used to map feature collections in arbitrary schemas to a target schema using a mapping description that is provided by the requester. • A service that implements both interfaces can be used in the same way. In this scenario, the requester does not necessarily have to provide the mapping description themselves but can query the Schema Mapping Service for an appropriate mapping description. • A service that implements the *SchemaMappingRepository* interface and another interface for creating or accessing feature collections (e.g. the interfaces of the Feature Access Service or the Processing Service) can be used to provide the output feature collections in different schemas.

Table 29: Description of the Schema Mapping Service

9.8.6 Ontology Access Service

Name	Ontology Access Service
Standard Specifications	The Ontology Access Service currently supports the following ontology language: • W3C OWL Web Ontology Language http://www.w3.org/TR/owl-features/
Description	The Ontology Access Service supports the read access to the specification of a logical ontology (see section 8.6.1.2) and to export or import a complete specification of a logical ontology into an ontology store. It provides a high-level view to the content of the ontology, allowing the client to get information about the taxonomy (classes and properties) defined by any stored ontology and to extract TBox and ABox vocabulary statements for human/machine interpretation. The Ontology Access Service is independent of any ontology technology, like the ontology language (e.g. OWL). However, the current version of the Ontology Access Service ignores ontological classes that are implicitly defined by rules of description logics (and only the explicit taxonomy is considered). Some typical usages of this service are: • Getting a list of the ontologies this service is providing access to; • Storing, updating or deleting available ontology entries; • Retrieving a partially or fully a stored ontology; • Getting high-level information about ontology, such as the list of concepts or the list of supported properties for a given concept and TBox (optionally ABox) Vocabulary statements. The Ontology Access Service provides the functionality through the following interfaces: • *ServiceCapabilities*: Informs the client about the common and specific capabilities of the Ontology Access Service. • *OntologyAccess*: Supports the storage, retrieval, and deletion of ontologies as well as providing a high-level view on ontologies. • *KnowledgeBase*: Optional interface providing operations to query and update models contained in the knowledge base (see section 9.6.5). • *TransactionInterface:* Optional interface providing update requests to a knowledge base in a transactional context (see section 9.6.4).

Interface *ServiceCapabilities*	
get Capabilities	Informs the client about the common and specific capabilities of an Ontology Access Service instance. Examples of specific capabilities are the names of the ontologies available at the servers and the supported ontologies (e.g. OWL). Examples of specific capabilities if the Knowledge Base Interface is being supported comprise: • The possible representation formats of query results which can be requested by clients. • The types of the models supported by the service that supports the Knowledge Base Interface (e.g. references to standards such as RDF, RDFS, OWL). • The query languages that can be used in knowledge base queries. • The inferencing capabilities of the knowledge base applied when computing query results.
Interface *OntologyAccess*	
parse Ontology	Given an ontology or a part (selection) of an ontology, it returns the hierarchy of classes (concepts) and properties that are defined by this ontology (a high-level view of the ontology). The format of the result could be, for example basic XHTML (without CSS) that is suitable for both direct display or further machine processing.
getTBox Vocabulary	Given an ontology or a part (selection) of an ontology, it returns a list of TBox statements ready to be used for creating a Knowledge Base.
getABox Vocabulary (optional)	Given an ontology or a part (selection) of an ontology, this optional operation returns a list of ABox statements ready to be used for creating a Knowledge Base.
setOntology	Stores a new ontology in the ontology store, if the ontology format is supported. The operation confirms the success of the operation by sending back to the client a Boolean “TRUE”.
getOntology	Retrieves an existing ontology or a part (selection) of an ontology from the ontology store.
Delete Ontology	Removes an existing ontology from the ontology store. The operation confirms the success of the operation by sending back to the client a Boolean “TRUE”.
Example usage	A party is having an ontology about forest fires and decides to share it with other parties. By invoking the *setOntology* operation, the ontology can be stored in the ontology store of the Ontology Access Service. The stored ontology can then be made accessible to other services. For example, using the *updateModel* operation of the the Knowledge Base Interface can use the ontology to expand the knowledge base with information about forest fires. Finally, if a client possesses an ontology and wants to present its structure directly on the Web, the *parseOntology* operation can be called giving the ontology as a parameter. The response is a high level-view of the ontology hierarchy, classes and properties that can be immediately displayed or it can be further processed. Additionally assuming that a client requires on ontology about forest fires and assuming that there are already some ontologies in the ontology store, a client shall call the *parseOntology* operation for each of the stored ontology in order to get a high-level view of the available ontologies and decide if one of the ontologies is adequate for the purpose. Then the client could retrieve the full ontology or only a part of the required forest fire ontology and pass it to other services for further processing tasks.
Comments	The following are out of the scope of the Ontology Access Service:

	• Creating ontologies – this service manages the storage and the access to ontologies, but doesn't provide any tool to create ontology structures. This is the purpose of dedicated tools (like Protégé) and methodologies (as the one defined in deliverable D2.3.2). • Inferencing – this is the responsibility of inferencign engine that may be attached to an implementation of the Ontology Access Service. • Remote editing of ontologies – it is assumed that the client, once it is getting the ontology from this service, will use specialized tools or API (like Protégé or Jena API) to deal with the ontology structure and editing. Calling operations on a remote service to work with ontologies doesn't seem reasonable in terms of architecture or usability. A high-level structure can however be provided for clients that do not need any details but just overall information about the ontology (using "parseOntology operation").

Table 30: Description of the Ontology Access Service

9.8.7 Thesaurus Access Service

Name	Thesaurus Access Service
Standard Specifications	• ISO-2788 Documentation -- Guidelines for the establishment and development of monolingual thesauri • ISO 5964:1985 Documentation - Guidelines for the establishment and development of multilingual thesauri.
Description	The Thesaurus Access Service supports read and write access to a thesaurus that may be multi-lingual. A thesaurus can be thought of as a synonym and antonym repository for data vocabulary terminology (Pollock, Hodgson 2004). As such, a thesaurus is a variant of an ontology restricting the relations used to a priori relationships between terms, e.g. questioning whether the meaning of two terms is similar, broader, or narrower. In a multi-lingual thesaurus these a priori relationships are not restricted to one natural language, e.g. a term A may be a synonym to term B even if term A is available in English and term B in French. The Thesaurus Access Service is a run time service that provides on-the-fly insight into data meaning by cross-referencing the included terms and providing a human readable description. In this capacity the Thesaurus Access Service provides crucial links in the resolution of unknown data semantics for requestors that are attempting to resolve new schema relationships in newly discovered models. The requestor may choose the language in which the terms requested shall be provided. The Thesaurus Access Service provides its functionality through the following interfaces: • *ServiceCapabilities:* Informs about the common and specific capabilities. • *ThesaurusAccessService:* Includes the operations for the read and write access to a thesaurus.
Interface *ServiceCapabilities*	
get Capabilites	Informs the requestor about the common and specific capabilities of a Thesaurus Access Service instance. Examples of specific capabilities are the supported languages and relations. Note: The reason to provide these capabilities is less to reflect the services logic capabilities than to reflect the available data.

Interface *ThesaurusAccessService*	
getScope	Gets a note attached to a term to indicate its meaning within an indexing language (i.e. a controlled set of terms selected from natural language and used to represent, in summary form, the subjects of documents; see ISO 2788).
getPreferred Term	Gets the preferred term when a choice between synonyms or quasi-synonyms exists.
getSynonyms	Gets the synonyms of a given term in a given language.
getAntonyms	Gets the antonyms of a given term in a given language.
getTopTerm	Gets the broadest class to which the specific concept belongs; sometimes used in the alphabetical section of a thesaurus (e.g. The concept African elephant would return animal in case of a biological thesaurus)
getBroader Term	Gets a concept having a wider meaning than the given term has.
getNarrower Terms	Gets a concept with a more specific meaning than the given term has.
getRelated Term	Gets an associated term, but that term is not a synonym, a quasi-synonym, a broader term or a narrower term.
setScope	Sets a note attached to a term to indicate its meaning within an indexing language
setPreferred Term	Sets the preferred term for another term
setSynonyms	Sets a synonym for a term in a given language.
setAntonyms	Sets an antonym for a given term in a given language.
setTopTerm	Sets the broadest class to which a term belongs
setBroader Term	Sets a broader term for a term.
setNarrower Terms	Sets a narrower term for a term.
setRelated Term	Sets an associated term for a term; that associated term is neither a narrower nor a broader nor a top term, nor is it a synonym, quasi synonym or antonym.
Example usage	An end-user can use the Thesaurus Access Service to determine synonym terms, which can subsequently be used to broaden a search.
Comments	none

Table 31: Description of the Thesaurus Access Service

9.8.8 Service Chain Access Service

Name	Service Chain Access Service
Standard Specifications	The standards related to the Service Chain Access Service are the following: • ISO 19119:2005 Geographic information – Services • OASIS Web Services Business Process Execution Language (WS-BPEL) http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel • XLANG - web services for business process design. Satish Thatte, Microsoft, 2001.

	• Web Services Flow Language (WSFL). Frank Leymann, IBM, 2001. • W3C Web Services Description Language (WSDL) 1.1 (http://www.w3.org/TR/wsdl) ISO 19119 defines the term “service chain” and in particular, the computational approach that is used in the Service Chain Access Service is based on the aggregate service pattern (opaque chaining). This is due to the fact that: i) current workflow engines support this approach, ii) the aggregate service pattern seems to be the more suitable one for the service oriented paradigm. WS-BPEL is the most credited language for expressing concrete service chains, others are XLANG and WSFL (WS-BPEL inherits all the main design constructs of both languages). Actually WS-BPEL is the only one which is continuously extended (now version 2.0 is available) and it is also the only one equipped with stable engines able to execute the service chain. Since service engines supply the execution of aggregated services by means of a single service, the WSDL standard is used to describe the corresponding interface. During the design of concrete service chain descriptions it could be necessary (e.g. in the case of very complex service chains) to start by using higher level languages for services choreography and then move to concrete and executable languages (e.g. WS-BPEL). The most credited standard for choreography is: • W3C Web Service Choreography Description Language (WS-CDL) 1.0. http://www.w3.org/TR/ws-cdl-10/
Description	The Service Chain Access Service supports the creation of an executable service instance based on an explicit description of a service chain. The chain can then be executed as a single service. However, the execution of the service is outside the scope of the Service Chain Access Service (see comment below). Based on the Reference Model of Open Distributed Processing (ISO/IEC 10746-1 RM-ODP) definition of chain of actions, a service chain is defined in ISO 19119 as a sequence of services in which, for each adjacent pair of services, occurrence of the first action is necessary for the occurrence of the second action. For the scope of this specification, it is important to distinguish between the description of a service chain (i.e. a document in some workflow language, e.g. BPEL), a deployed instance of a chain (i.e. an executable piece of code), and the actual process of executing the chain. The service specification is based on the aggregate service pattern where services appear as a single service which handles all coordination of the individual services that are part of the chain. The *createServiceChain* operation supports a service provider in creating an executable instance of an aggregate service based on an explicit service chain description, and optionally registering that service instance with a catalogue service. The Service Chain Access Service provides its functionality through the following interfaces: • *ServiceCapabilities:* Informs about the common and specific capabilities. • *ServiceChainAccessService*: Selection of service chain descriptions and creation and deletion of aggregate services based on such descriptions.
Interface *ServiceCapabilities*	
get Capabilites	Informs the requestor about the common and specific capabilities of a Service Chain Access Service. An examples of a specific capability is the supported workflow language in which the service chain description can be specified
Interface *ServiceChainAccessService*	

createService Chain	Deploys the service chain instance (an aggregated service) specified in a workflow document
getService Chain	Gets a descriptor of the service chain which includes meta-information (id, address, description, and workflow language) and the workflow description itself.
deleteService Chain	Deletes a service chain instance.
Example usage	A client creates an aggregate service which can access features and perform schema transformations. This service can now be accessed as one single service from a client.
Comments	In a service implementation the *Service Chain Access Service* and *Processing Service* interfaces can be combined. The workflow language can then be used to define combinations of several processing operations of this service instance. Thus, a combination of related processing operations can be executed with one call without having to send the same data repeatedly to the service.

Table 32: Description of the Service Chain Access Service

9.9 OT Support Services

Note: Some of the OT Support Services do not (yet) comprise descriptions of the service operations as the functionality of these services still needs further discussion within the ORCHESTRA project. The result of this discussion will include the list of OA Services and other OT Support Services that may be used by a given OT Support Service in order to provide its functionality according to the functional classification of the ORCHESTRA Services (see section 9.3).

9.9.1 Processing Service

Name	Processing Service
Standard Specifications	The functionality of the Processing Service is based on the WPS OGC draft implementation specifications: • OGC 05-007r4 Web Processing Service (WPS), version 0.4.0 (discussion paper) The interface of the WPS is that of a general purpose Web Processing Service which provides client access to pre-programmed calculations and/or computation models operating on spatially referenced data. Access happens through one generic execute operation that initiates a process based on a number of input parameter values and an output definition. The WPS concept of a single execute operation has been adopted as is for the Processing Service. The processing may occur on features which can then be encoded according to given standards (see the Feature Access Service in section 9.7.1).
Description	The Processing Service describes a common interface for services offering processing operations on spatial (vector as well as raster) and non-spatial data. Examples of processing operations are statistical or geospatial calculations, image processing and analysis or, in general, computer algebra operations. The Processing Service provides mechanisms to identify the data required by the calculation, initiate the calculation, and manage the output so that it can be accessed by the client. The Processing Service provides its functionality through the following interface: • *ServiceCapabilities*: Informs about the common and specific capabilities. • *ProcessingService*: provides the means to get information on and to invoke a

	specific processing operation.
Interface *ServiceCapabilities*	
get Capabilities	Informs the requestor about the common and specific capabilities of a Processing Service instance. Examples of specific capabilities are the supported processing operations (name and abstract).
Interface *ProcessingService*	
getProcess Description	This operation allows a client to request and receive detailed information about one or more processing operation(s) that can be executed by an execute operation, including the input parameters and formats, and the outputs.
execute	This operation allows a client to execute a specified processing operation implemented by the Processing Service, using provided input parameter values and returning the outputs produced.
Example usage	A client wants to create a buffer zone around a forest during a fire and calculate the total area that is included in the buffer. The client queries the Processing Service for a description of the buffer processing operation (including its input and output types) using the *getProcessDescription* operation and then calls the buffer processing operation using the execute operation. The Processing Service returns the result of the buffer processing operation either directly or as a reference (that can be used by the client to access the result).
Comments	In order to avoid having to send the same data repeatedly to the same instance of a processing service to execute several related operations, it should be possible to invoke a combination of related processing operations with one call to the service. This can be achieved by a service instance that implements both the Processing Service and the Service Chain Access Service (SCAS) interface. Thus, a SCAS workflow language can be used to define combinations of processing operations. The optimisation of “local” operation calls is an issue that should be addressed at the implementation level. For the implementation of GIS functionalities, several (Open Source) GIS libraries exists, both for vector and raster data processing: • GRASS *http://mpa.itc.it/markus/grass50progman/node98.html*, (including OGC-conformal (Open Geospatial Consortium) Simple Features for interoperability with other GIS) • Terralib *http://www.terralib.org/* • GeoTools *http://www.geotools.org/display/GEOTOOLS/Overview* • GMT *http://gmt.soest.hawaii.edu/* • Map window *http://www.mapwindow.com/* • OpenEV *http://openev.sourceforge.net/* • Jump *http://www.jump-project.org/* • STARS: Space-Time Analysis of Regional Systems, *http://stars-py.sourceforge.net/whatisstars.html* For the implementation of statistical functionalities, many tools and libraries are available. The mathematical algorithms used by the service operations could be taken from existing libraries, e.g: • OCTAVE *http://www.gnu.org/software/octave/* (Free, Opensource) • Statistical analysis libraries such as R (*http://www.r-project.org/*) or Matlab (*http://www.mathworks.com*). • List of free software available at *http://members.aol.com/johnp71/javasta2.html*

	• A complete Statistical Analysis Software Survey available at http://www.lionhrtpub.com/orms/surveys/sa/sa1.html • See http://mathworld.wolfram.com/ for terminology and operator explanation An alternative architectural approach could be taken such that no Processing Service interface is described on the abstract level. Instead, the OMM would contain detailed rules about how processing service interfaces may be described by service providers. These descriptions should then include a process description, the input and output of the service and binding information, i.e. all information that is currently described in the Processing Service's *getProcessDescription* operation. In both cases and for a common understanding of processing operations, (basic) operations should be grouped and described in an operation taxonomy to be referenced in the service specific capabilities. Guidelines could be e.g. the Map Algebra operations (Tomlin 1990) or the Egenhofer Operators (Egenhofer 1989).

Table 33: Description of the Processing Service

9.9.2 Simulation Management Services

Name	Simulation Management Service
Standard Specifications	• OGC 05-007r4 Web Processing Service (WPS), version 0.4.0 (discussion paper)
Description	The Simulation Management Service allows the user to discover, specify input for, and control execution of a variety of simulation models. A simulation could be anything from a simple service which combines two numbers to a large simulation based on complicated mathematical models predicting the weather. The Simulation Management Service allows the implementer to allow others to discover, execute and control their model in a simple and generic fashion. The Simulation Management Service allows the model to initially support multiple simulations (which also could be derivatives of a particular model). The user can then ascertain the specifics of what the model requires to run (including additional input services and a description of the parameters required). The Simulation Management Service then provides the user the ability to execute and check on the models progress. They can also modify the currently executing model to dynamically modify the scenario. The Simulation Management Service provides its functionality through the following interfaces: • *ServiceCapabilities:* Informs about the common and specific capabilities. • *AsynchronousInteraction:* Exploits the OA Basic Service to provide a mechanism to invoke a simulation and obtain an ID for the simulation such that subsequent modification and query requests for that simulation can be made. • *ProcessingService:* Provides the operation to call the simulation run. • *SimulationManager:* Provides the interface to describe in detail the inputs required to invoke a supported simulation, as well as its outputs. The interface also provides operations to modify, suspend or resume an executing simulation, and to query its status.
Interface *ServiceCapabilities*	

getCapabilities	Informs the requestor about the common and specific capabilities of a Simulation Management Service instance. Examples of specific capabilities are the abilities of the simulation manager to include the types and versions of simulations supported by the simulation service
Interface *AsynchronousInteraction*	
invokeAsync	Starts asynchronous execution of a simulation. The *invokeAsync* operation returns immediately with an identifier (invocation ID) representing the asynchronous execution. In order to receive notifications a reference to a callback interface can be provided.
abort	Aborts execution of a simulation identified by its invocation ID.
notify	Passes a notification to the callback interface provider (to be implemented by the SimMS client).
Interface *ProcessingService*	
getProcess Description	Requests and receives detailed information about one or more processing operation(s) that can be executed by an execute operation, including the input parameters and formats, and the outputs.
execute	Executes a specified processing operation implemented by the Processing Service, using provided input parameter values and returning the outputs produced.
Interface *SimulationManager*	
modify Process	Applies a change to one or more simulation parameters during the execution of a simulation, to take effect from a defined point within the simulation. The simulation to be modified is identified by its invocation ID obtained by the *invokeAsync* operation. This operation also allows requests to the simulation state to be made to either suspend or resume execution.
query Process	Queries the state of a simulation identified by its invocation ID, to determine information such as whether the simulation has been suspended, is executing or has completed. As an option, this operation also provides the percentage complete.
Example usage	The caller wishes to execute a model. – Through *getCapabilities* the caller can discover what simulations can be executed. – On choosing a particular simulation the caller can then invoke *describeProcess* which reveals the requirements of the simulation. – The simulation is then invoked by *invokeAsync* which will execute the simulation. If the input to the simulation is ill-formed or invalid the execution will be aborted and the caller will have to re-specify. – The calling system can poll via *queryProcess* to find out the status of the simulation. – The caller may make dynamic modifications of the active scenario via *modifyProcess* (e.g. moving the position of a spill or adding extra wind). – When the simulation has completed, the SimMS returns the simulation results through the client's notify operation.
Comments	none

Table 34: Description of the Simulation Management Service

9.9.3 Sensor Planning Service

Name	Sensor Planning Service
Standard Specifications	• NASA/JPL Sensor Webs Project (http://sensorwebs.jpl.nasa.gov/). • OGC 05-086r2 - Sensor Model Language (SensorML) Implementation Specification V1.0 (Draft proposed version) • OGC 05-089r3 – Sensor Planning Service Implementation Specification V0.0.30 (Request for Comments)
Description	Following the OGC Sensor Planning Service Discussion Paper: "The Sensor Planning Service 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. Not only must different kinds of assets with differing capabilities be supported, but also different kinds of request processing systems, which may or may not provide access to the different stages of planning, scheduling, tasking, collection, processing, archiving, and distribution of requests and the resulting observation data and information that is the result of the requests. The Sensor Planning Service is designed to be flexible enough to handle such a wide variety of configurations."
Example usage	A client wants to gather a satellite scene of a certain sensor for a certain region. The Sensor Planning Service offers the client a way to define the required parameters and to set up the respective notification mechanisms.
Comments	The specification of this service shall be aligned to the ongoing specification work within the OGC working group dealing with "Sensor Web Enablement".

Table 35: Description of the Sensor Planning Service

9.9.4 Project Management Support Service

Name	Project Management Support Service
Standard Specifications	• ISO 10006:2003 Quality management systems -- Guidelines for quality management in projects • ISO 10007:2003 Quality management systems -- Guidelines for configuration management • PMI Project Management Body of Knowledge (PMBOK) (http://www.pmi.org/) • Project Management XML Schema (PMXML) (http://xml.coverpages.org/projectManageSchema.html) • dotProject - the Open Source Project Management tool (http://www.dotproject.net/index.php)
Description	The Project Management Support Service supports the planning and performance of operations (projects) in a cooperative distributed environment in cases where a desktop project management tool is not sufficient. Its purpose is to specify a project based on definitions according to the following dimensions of project management: - the structure of a project into project elements, i.e. the division of a project into sub-projects, work packages and tasks, the identification of logical dependencies between the project elements, the assignment of costs and priorities to the project elements and the identification of project results and partial results. - the structure of the resources, i.e. the identification of the type and number of

	resources (human resources, organisation units, machines, tools, computation resources, network bandwidth, ORCHESTRA features, ORCHESTRA services, meeting resources…), their characteristics (e.g. competences in case of human resources), their relationships (e.g. tool is part of a machine, person belongs to a organisation unit) and their location. - the time horizon, structured into units of, for example, months, weeks, days, hours, minutes in accordance with the plan horizon and the level of plan detail. Time oriented attributes include start and end dates of project elements, the identification of milestones and delivery dates for project results, the time dependencies between project results, the (estimated and actual) duration of project elements and the availability of resources during a given plan horizon. - the spatial dimension describing the location and movement of resources and where the project elements are to be executed. This service comprises the operations in the following operation groups: - to specify the project according to the three dimensions illustrated above with a close interlink to resources in an OSN. - to support queries about a project, like e.g. “Which resources are assigned to which task ?”, “What is the pre-requisite to deliver project result A ?”, “Which document is required to carry out task B ?” - to specify and optimise the allocation of resources to different tasks based on, for example, their importance, their order in which they must be undertaken and competition for the same resources. - to optimise the timely delivery and to calculate and optimise the cost of the project results - to specify and evaluate project scenarios based on multi-criteria optimisations The Project Management Support Service provides the following capabilities: list of supported project management techniques and their options, list of supported operations structured according to operation groups
Example usage	The service may be used in the risk management domain to support the development and evaluation of emergency plans in case of a natural hazard in a given area, e.g. the evacuation of a settlement in case of a threatening forest fire.
Comments	The service operations are based upon known project management techniques such as Gantt diagrams, PERT (Program Evaluation and Review Technique), CPM (Critical Path Method), PSP (Project Structure Plans) or Critical Chain Method. The applicability of more recent techniques such as that of the Business Communication Engineering tool Communigram® will be investigated (http://www.communigram.com/).

Table 36: Description of the Project Management Support Service

9.9.5 Communication Service

Name	Communication Service
Standard Specifications	• IETF 3261 SIP: Session Initiation Protocol, June 2002 • ITU T.120 Data protocols for multimedia conferencing • ITU H.323 Packet-based multimedia communications systems • OGC 03-029 OWS Messaging Framework (OMF) V0.0.3
Description	The objective of the Communication Service is to provide harmonised access to di-

	rect user-to-user communication means based on multi-media technologies and data exchange between users. Harmonised access is required as these services are most often associated with collaboration within a user community according to a common community objective (e.g. a project) which is not supported by the existing tools and standards in a common approach. The service will directly support users and provide them with the support to conduct interactive collaboration. Examples include: - Presence Awareness: ability to determine who is on line at a given instant - Chat: ability for multiple users to type text data onto their local device and the text can be seen by other chat session participants - Instant Messaging: combining Presence Awareness and Chat - Polling / Surveying: providing the ability for a user to request a vote from other collaboration participants - White boards: to interactively manipulate graphical objects with other users - Application Sharing / Desktop Sharing / File Sharing: provides users with the ability to control a shared application remaining running on the sharers computer (for example to allow multiple users to update a single document interactively) - Shared Storage: provides multiple users with a common place to upload and download files - File Transfer: to transfer a file to another user or set of users - Shared Calendars / Scheduling: provides a group of users with a common calendar that all may directly interact with - Teleconference (audio and/or video) - Audio and/or Video Broadcast The Communication Service indicates the following capabilities to the requestor: the interactive collaboration services supported together with the operations and options related to each of them.
Example usage	Usage through OA Services e.g. 1. Building of user communities and assigning access rights or 2. News registration and communication service Potential uses of collaborative communication services include, e-learning, workflow management, decision support, mission planning and logistics.
Comments	It is to be decided if parts, at least, of this service are better classified as Human Interaction Components than as Workflow/Task Management Services. The component could be a community portal integrating different communication services like e-mail, newsgroups or Internet Relay Chat.

Table 37: Description of the Communication Service

9.9.6 Calendar Service

Name	Calendar Service
Standard Specifications	• ISO 8601: 2004 Data elements and interchange formats -- Information interchange -- Representation of dates and times • ISO 19108:2002 Geographic information - Temporal schema.
Description	The Calendar Service performs arithmetical date/time functions, comparisons and format conversions. As most information in thematic domains has a temporal dimension with a reference to a calendar date (e.g. a measurement value), there is a need to support calculations using these dates (e.g. for time series analysis in case of measurement series). The service provides operations to convert between different representations and

	the usual one using year, month, day, hour, minute and second, - to compare two dates and to perform simple arithmetical functions like adding/subtracting a number of days or seconds and computing the difference between two dates, - to create a calendar for any month, past, present and future, for easy use with other services, - to perform calculations between dates, reducing time computations to simple arithmetic. The Calendar Service indicates the following capabilities to the requestor: list of operations supported, including the parameters and their expected format
Example usage	To try to recreate history or project the future one might need to know just what day was the first Sunday of November 1963 or what day of the week May 12, 2034 will be. The service allows a client to enter a date, to specify a number of days to be added (to check a future date) or subtracted from (to check a past date) and to get the new date. Or, it allows a client to specify a pair of dates in order to calculate the number of days between these.
Comments	none

Table 38: Description of the Calendar Service

9.9.7 Reporting Service

Name	Reporting Service
Standard Specifications	• OASIS Open Document Format for Office Applications (OpenDocument) (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office)
Description	The Reporting Service supports the creation of reports using actual information from other services according to a given template. The process to create a report can be of very high complexity. Thus, instead of providing a generic report generator, this service offers a wrapper interface to existing products and tools for report generation. While many report formats are imaginable, for practical reasons only standardised formats are supported.
Example usage	The result of a seismic risk assessment has to be publicised regularly in a format that has been standardised by a civil protection agency. The Reporting Service supports this task by allowing a template to be provided once according to the report standard and filling the template based on the actual data.
Comments	For reporting there might be more than one source for input data. For simple reports a configurable service may be provided, for special cases subclasses of this service can be created.

Table 39: Description of the Reporting Service

9.10 OA Service Interaction Patterns

The combined usage of the OA Services and the ORCHESTRA Information Models is illustrated by means of OA Service interaction patterns. Note that these interaction patterns are informative and just provide examples. It is not claimed that this is the only way of using and combining the OA Services nor that this way is complete.

The following OA service patterns are currently described:

- Controlled user access to resources
- Rights-managed user access to resources
- Integration of source system data into an OSN
- Registration of resources in a catalogue
- Generation of meta-information
- Semantic catalogue component

9.10.1 Controlled User Access to Resources

9.10.1.1 Overview

The controlled user access to resources is described by an interaction pattern that involves the User Management, the Authorisation and the Authentication Service (UAA services). This pattern assumes the following context and OSN characteristics:

- Two departments of one organisation are attached to the same OSN and share a common UAA policy (see section 11.1.5).
- The OSN comprises OSIs of a Format Conversion Service, a Document Access Service and a Feature Access Service that use one User Management OSI, one Authentication OSI and one Authorisation OSI in the following way:
  - The Format Conversion OSI is owned by Department 1. The Feature Access OSI and the Document Access OSI belong to Department 2.
  - Department 1 and 2 have administrators “admin 1” and “admin2”, respectively.
  - Each administrator is responsible for the services of his department.
  - Department 1 and 2 have employees “user 1” and “user 2”, respectively.
  - The Authentication OSI implements a username/password authentication mechanism.
  - The Authorisation OSI implements a role based authorisation paradigm.

9.10.1.2 Scenario “UAA Setup”

This scenario cannot be described in detail because the setup procedure of each service depends on its implementation. Nevertheless, we can describe in principle how such a setup could look.

1. The Authentication OSI is set up. During the setup the first principal called root principal is created. The root principal can be authenticated and the resulting session information is used during the setup of the Authorisation and User Management OSIs.
2. Each UAA service OSI has a simple built-in authorisation component which grants all available permissions to the root principal until the actual Authorisation OSI has been configured.
3. The root principal creates the admin principals “admin 1” and “admin 2”.
4. The next step is to register the User Management and Authentication OSIs as well as the Feature Access, the Format Conversion and the Document Access OSIs in the Authorisation Service. How this is done is specific to the Authorisation Service implementation.
5. After the Services have been registered the root principal creates admin permission for

principals “admin 1” and “admin 2” for the corresponding services.

6. For security reasons the root principal will be deactivated.

From now on “admin 1” and “admin 2” are able to administer their services.

9.10.1.3 Scenario “Create new User”

1. The human “John Doe” behind “user 1” has demanded a user account.
2. Admin1 creates a principal “user 1” in the Authentication OSI.
3. Admin1 creates a subject with John Doe’s personal information as subject attributes in the User Management OSI.
4. Admin1 assigns principal “user1” the newly created subject using the *addPrincipaltoSubject()* operation of the User Management OSI.

User1 is now a valid system user but cannot access any service due to the lack of corresponding service permissions.

9.10.1.4 Scenario “Permission Assignment”

1. “User 1” has requested permissions to access the Format Conversion OSI.
2. “Admin 1” assigns an operation permissions for the *convert* operation of the Format Conversion Service to the principal “user1”.

“User 1” is now able to able to invoke the Format Conversion Service.

9.10.1.5 Scenario “Service Request”

1. “User 1” wants to invoke operation *convert* against the Format Conversion OSI.
2. In order to receive session information “user 1” (the client software of “user 1” respectively) uses the Authentication OSI to authenticate his “user 1” principal using his password.
3. “User 1” attaches the session information to the *convert* operation of the Format Conversion Service.
4. The Format Conversion OSI parses the session information and extracts the reference to the Authentication OSI of the authenticated principal(s).
5. The Format Conversion OSI makes a request to the Authentication OSI to verify session information. Verification of session information is implementation specific and might use session keys, for example.
6. The Format Conversion OSI creates an authorisation context and passes it to the *authorise* operation of the Authorisation Service. The structure of the authorisation context is known to the application and specific to the permission types supported by the Authorisation Service. For a operation permission type, for example, the operation context includes the name of the operation to be invoked.
7. The Authorisation OSI receives the authorisation context. It checks whether the given principal (included in the authorisation context) has sufficient permission to invoke the requested operation. This is done within an implementation and permission type-specific decision process. Evaluating an operation permission means, for instance, to check whether the given operation may be invoked. An evaluation of a time coverage permission might require a comparison between the current timestamp and a time coverage given in the permission associated with the current principal.
8. The Authorisation OSI returns a compliance value representing the authorisation decision.
9. The Format Conversion OSI interprets the compliance value. It throws an *OA_PermissionDeniedException* for a negative compliance value and performs the operation for a positive one.

9.10.2 Rights-Managed User Access to Resources

Rights-managed user access is an extension of the controlled user access described above. It is based on two different preconditions:

- The user or the according user group needs access rights to the resource which are stored in the Authorisation service
- The user needs to either recognise or explicitly agree to a license agreement including usage constraints, based on a Creative Commons[5] approach.

The OGC GeoDRM Reference Model (GeoDRM, 2006) defines a right as a "permission to act that makes principal entitled to act with respect to all or part of a specified resource for a certain period of time", whereas a license is defined as a "representation of grants that convey to principals the rights to use specified resources subject to specified conditions".

An example of a Geo Rights Management implementation architecture based on the Authorisation Service (see section 9.7.8) and Authentication Service (see section 9.7.9) is shown in

Figure 35.

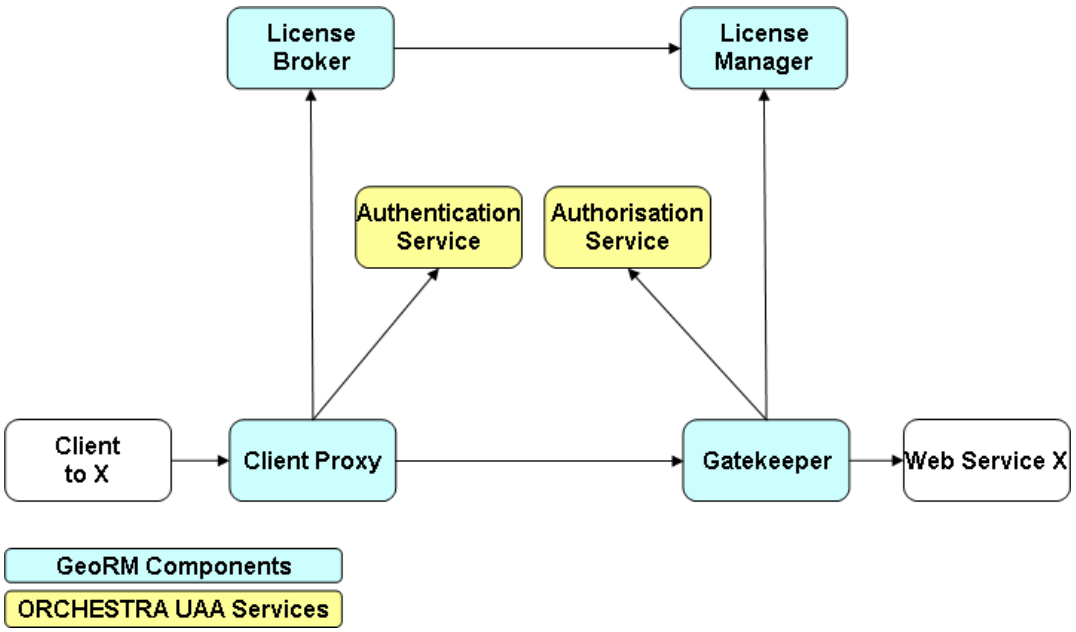

Figure 35: Service Interaction Pattern for Geo Rights Management

This architecture assumes that there is a Web service X (e.g. an OSI or an OGC service instance) and an corresponding client, both not capable of providing GeoRM functionality. These two components are wrapped with proxy components on the client side (Client Proxy) as well as on the service side (Gatekeeper).

The Gatekeeper adds a license code to the service capabilities of X. A license code may contain

- usage restrictions (similar to Creative Commons),
- preconditions (e. g. authentication required, license acceptance required, …) and/or
- obligations (e. g. time-limited access, restricted number of requests, …).

Service requests for the Web service X are first received by the Gatekeeper which checks if the pre-conditions are fulfilled (e. g. if a valid identity or license token is provided) and if the requesting identity is permitted to perform that request, i.e. the request has to be covered by the license and must be al-

[5] http://www.creative-commons.org

lowed by the policies stored in the Authorisation Service.

If this is the case, the Gatekeeper applies the obligations (if existent, e.g. by counting requests and blocking after the maximum number of requests is reached) and forwards the request to the service. The service then produces the requested result, which is conveyed back to the client.

The Client Proxy is responsible for interpreting the license preconditions and obtaining the required tokens, either an identity or a license token.

In order to get an identity token, the Client Proxy accesses the Authentication Service with the credentials of the principal that represents the user or the client software component (see section 7.5.2). The Authentication Service checks the authenticity of the principal and returns the identity token (including an authentication statement) if the check has been successful.

In order to get an identity token, the Client Proxy accesses the License Broker. The License broker returns a license token that conatins a reference to the original license stored in the License Manager.

If the Gatekeeper receives a request including an identity or a license token, it delegates the policy decision to the Authorisation Service by calling an *authorise* operation. Furthermore, it obtains the license referenced in the license token from the license manager. If the authorisation process is successful and the license covers the current request, then the policy decision is set to “permit”. If there are any obligations included in this license they are transmitted to the Gatekeeper as well. Thus, the Gatekeeper is enabled to enforce the policy decision in its entirety.

9.10.3 Integration of Source Systems into an OSN

Source System Integration has been defined in section 7.6 as the process of transforming an External Source System into an ORCHESTRA Source System. Thus, it starts in a native (i.e. non-ORCHESTRA) environment and results in a running OSI that represents the access point to the data and functionality of an External Source System within an OSN. This OSI must be built according to the rules that are defined in the ORCHESTRA Service Meta-model (OMM-Service as described in section 9.2).

Integration of one or more External Source Systems into an OSN means creating (at least) one new OSI. This instance is created by defining and implementing an ORCHESTRA conformant interface resulting in a service that is able to interact with the External Source System. For the description of this OA pattern, the resulting OSI is called Source System Integration Service in order to have a single name for the entirety of services of this kind. It is a surrogate name since Source System Integration Services needn’t share any predefined interface type (apart from the mandatory *ServiceCapabilities* interface described in section 9.6.1) that could be used as a name instead.

Note: The name Source System Integration Service neither states that any specific interface is implemented, nor does it create a new service type, since a Source System Integration Service might as well be just an implementation of the service type Feature Access Service.

Starting on an abstract level, the integration process of source systems can be described in the following steps:

1. Check the available interfaces types of the defined ORCHESTRA Service Types and select (if any) the interfaces that are suitable to represent the External Source System. (e.g.: a database might be best represented through a *FeatureAccessService* interface as specified in section 9.7.1). This step is not restricted to selecting only one interface type, therefore it’s valid for a Source System Integration Service to realise multiple interface types as defined in abstract specifications.
   According to the OMM-Service, at least the *ServiceCapabilities* interface must be selected in this step. If the External Source System provides operations that the integrated ORCHESTRA Source System shall offer to the OSN, continue with step 2. If there aren’t any further operations continue with step 3.
2. If the collection of selected interface types does not completely fit a predefined ORCHESTRA Service Type, a new service type shall be defined.
3. There are two possible ways to integrate any operations that the External Source System provides. One of these must be chosen as illustrated in Figure 36.
   a. Extend the Source System Integration Service’s interface with a new operation for

every operation of the External Source System that should be integrated (and therefore visible in the OSN).

b. Implement the *SynchronousInvocation* interface (see section 9.6.2) and add the external operations as possible parameters to the *invoke()* operation.

4. Transform the native meta-information that will be needed within the OSN into ORCHESTRA meta-information according to the rules defined in Annex B1 of the RM-OA (or define such meta-information from scratch if it is not available yet). A non-exhaustive list of such meta-information that would be the contents of the service capabilities (e.g.: provider information, interface self-description…), the OAS, the feature type descriptions, ontologies, parameter types…

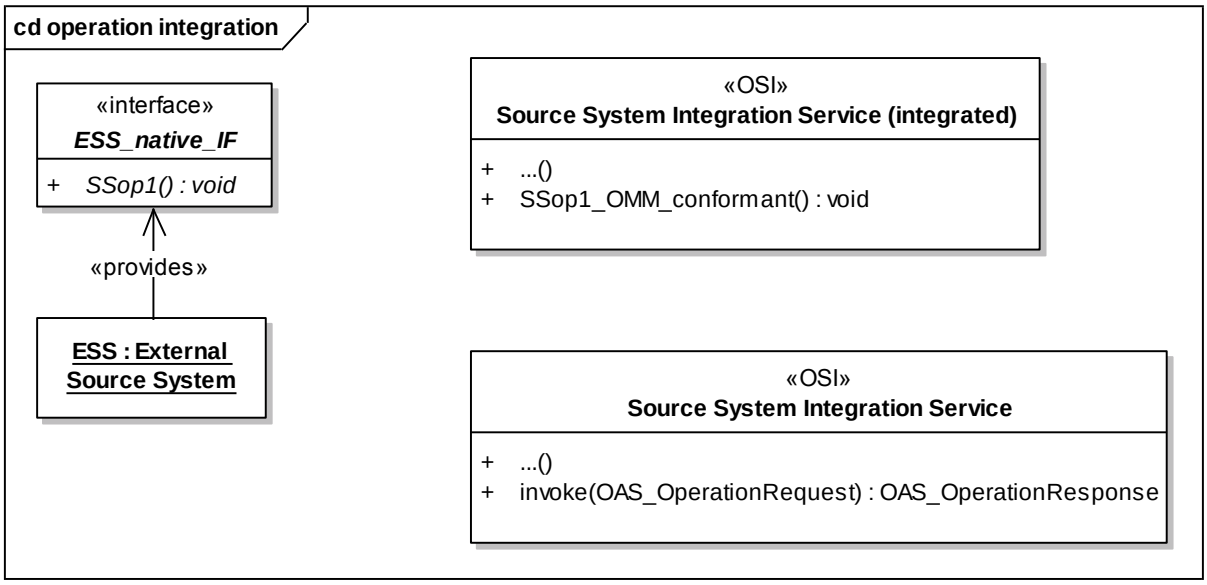

Figure 36: Operation Integration (upper right: SSI step 2a, lower right: SSI step 2b)

5. For a given platform, provide an implementation specification for the interface types of the Source System Integration Service.
6. Develop an OSC that corresponds to the implementation specification. This can be done either by mapping the interface operations to the native interface operation of the External Source System or by implementing the functionality from scratch.
7. Create and start an instance of the Source System Integration Service (a respective OSI) within the OSN.

Note 1: These steps are the tasks a source system provider must perform in order to integrate his External Source System into an OSN when starting on abstract level. Of course, these steps can be supported by tools in order to result in a (semi-) automatic integration process.

Note 2: A corresponding integration process could be defined when directly starting on platform-specific level.

Note 3: During all of those steps existing interface types of OA/OT-Services and also implementations of OA/OT-Services might be used to facilitate the tasks that need to be performed in the integration process (e.g.: a Schema Mapping Service might be used to transform a database schema into an OAS). But implementations of the OA/OT-Services are not required to support the integration process in any way, since this would mean that those services have to operate outside the specified boundaries of ORCHESTRA.

Figure 37 shows the basic and common interfaces among all integrated source system integration services. Since the type of the External Source System is unknown, it is impossible to know the interfaces needed for all possible External Source Systems. Therefore a generic and also extendable interface must be given as a base.

All that is predefined is the required service self-describing operation *getCapabilities()*.

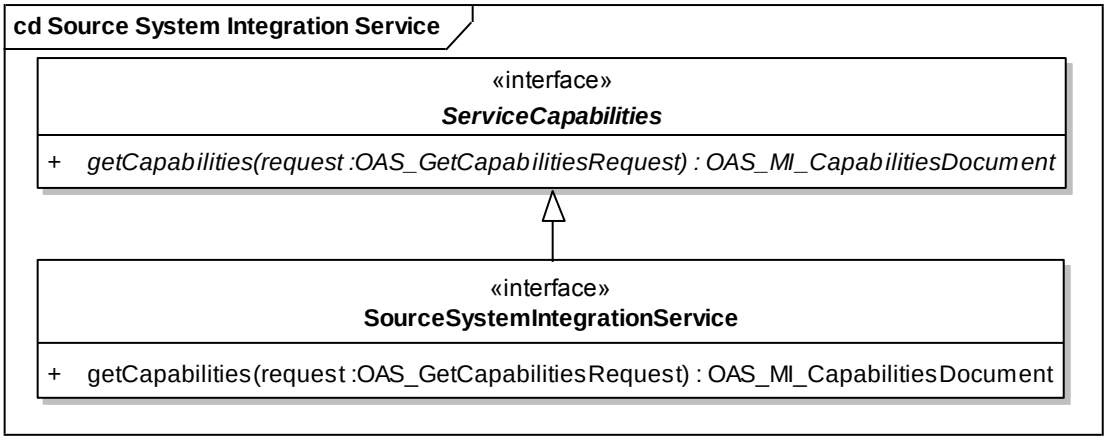

Figure 37: Source System Integration Service

In order to be able to support the wide heterogeneity of available External Source Systems, the Interface of the Source System Integration Service can be extended as the integrator desires. This includes inheriting and implementing interfaces of predefined OT/OA-Services as described in the RM-OA as well as adding new operations unrelated to any predefined interface type. Of course the meta-information, especially the interface description in the service capabilities, must reflect this. Thus, it contains all operations that are available at the service, having in mind that there might not be a hand-written specification of the service in case of a fully automated source system integration process.

9.10.4 Generation of Meta-information

Several OA Services provide the means for the generation of meta-information. Figure 38 outlines known methods for that purpose and assigns the respective OA Service to each method.

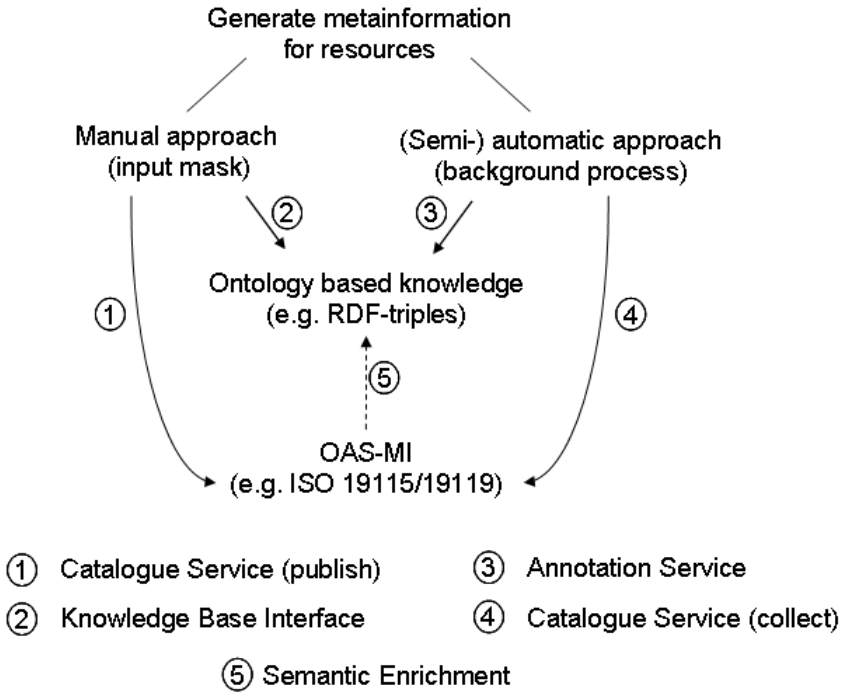

Figure 38: Generation of resource meta-information

Meta-information is generated for various types of resources, being feature or service instances, according to a well-defined purpose (see section 8.4). The main criteria for the classification of methods are the distinction between manual and automatic (or semi-automatic) approaches.

Manual generation of meta-information is usually carried out by a human user, who inserts values into certain fields of meta-information of an input mask. On the one hand, meta-information may consist of simple attributes, such as keywords for discovery purposes, which can be used to find resources by applying a boolean match. The attributes may then be structured according to an application schema for meta-information (OAS-MI, see section 8.4). Concrete examples are meta-information standard such as Dublin Core or ISO 19115, or ISO 19119 in case of service meta-information. The Catalogue Service (see section 9.7.5) can be used for the publication of meta-information for discovery purposes. The publication can be performed manually using a GUI-based client component that accesses the *CataloguePublicationInterface* (see method ① in Figure 38) or semi-automatically by means of the *CatalogueCollectionInterface* (see method ④ in Figure 38). These two alternatives are described in more details in section 0 below.

A more advanced method for describing resources is to edit statements which can be added to a knowledge base by means of operations of the Knowledge Base Interface (see section 9.6.5), where they are stored as a knowledge graph. An implementation example of such a knowledge base is an RDF (Resource Description Framework) Triple Store. The statements describe the relationship from resources to concepts of an ontology and their relationship to other resources as well. Thus, this kind of meta-information is on a semantic level, as it can be interpreted by an ontology. The Knowledge Base Interface (see section 9.6.5) allows a user to manually update the knowledge base (see method ② in Figure 38).

However, the OA also aims at supporting an automatic approach by means of the Annotation Service (see method ③ in Figure 38 and the service description in section 9.8.3). The Annotation Service identifies named entities in texts and automatically establishes relationships between resources and concepts and between resources among each other. The information in such a knowledge base can be explored by browsing the ontology using dedicated navigation tools or by formulating exact queries in an ontology query language.

As a future possibility, currently not supported by an OA Service, is the possibility to enrich a meta-information schema by references to concepts of an ontology, illustrated as method ⑤ in Figure 38.

9.10.5 Registration of Resources in a Catalogue

Registration means the creation of an associated meta-information entry of a resource (data or service) in a catalogue in order that a user in an OSN may discover the resource. The registration of the resources can be achieved via the *CataloguePublicationInterface* and the *CatalogueCollectionInterface* of the Catalogue Service (see section 9.7.5), which provides means for including, updating and deleting catalogue entries. The *CataloguePublicationInterface* provides a push paradigm and the *CatalogueCollectionInterface* provides a pull paradigm.

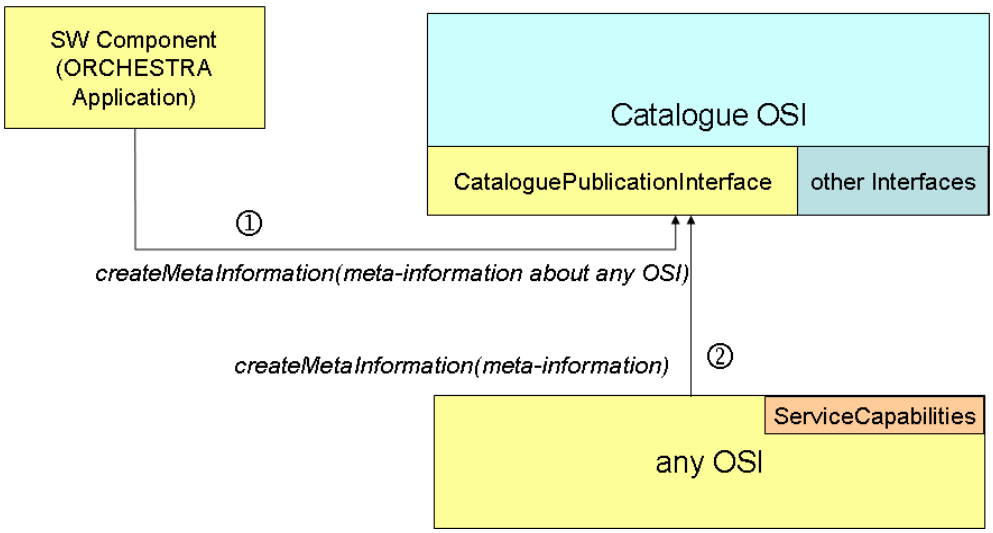

Figure 39: Generation of meta-information entries (push paradigm)

The meta-information in a catalogue is structured according to an OAS-MI that the catalogue is able to

handle. The following figures illustrate an example in which an OAS-MI is structured according to the capabilities of ORCHESTRA services which need to be described in the catalogue. In this example, the meta-information is extracted from an OSI by calling the operation *getCapabilities()* contained as part of the *ServiceCapabilities* interface of any OSI.

Now, the push paradigm is supported by the operations *createMetaInformation()* and *setMetaInformation()*. By calling these operations software components of ORCHESTRA Applications (see case ① in Figure 39) or any OSI itself (see case ② in Figure 39) can directly store meta-information in the catalogue.

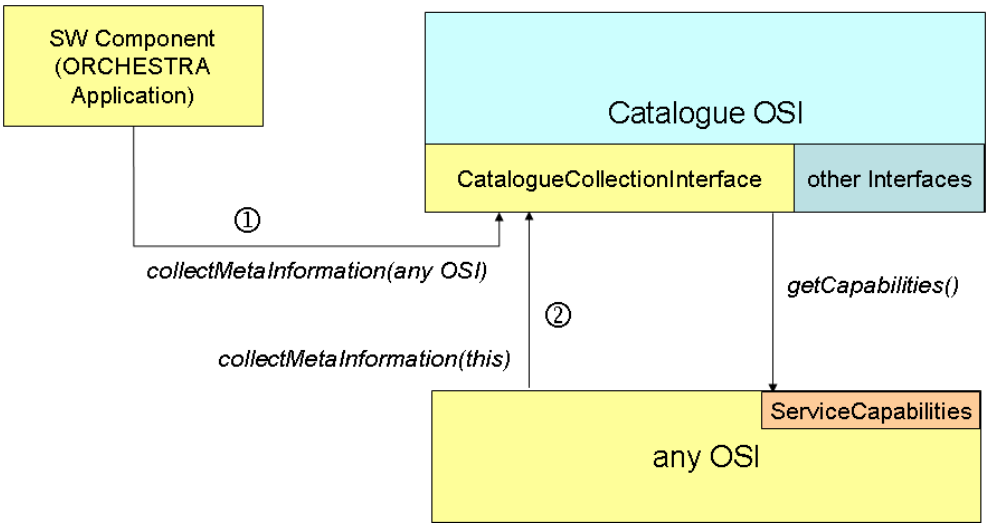

Figure 40: Generation of meta-information entries (pull paradigm)

The pull paradigm is supported by the operations *collectMetaInformation()* and *collectMetaInformationPeriodic()*. By calling these operations software components of ORCHESTRA Applications can trigger the catalogue to pull meta-information from an OSI (see case ① in Figure 40) or an OSI itself can trigger the catalogue to pull the meta-information (see case ② in Figure 40). *CollectMetaInformation()* is used for a single pull, while *collectMetaInformationPeriodic()* is used for periodic updates of the resources.

9.10.6 Semantic Catalogue Component

An OSC called “Semantic Catalogue” may be built by combining various instances of the ORCHESTRA Catalogue Service (see section 9.7.5) and the Ontology Access Service (see section 9.8.6) in a service interaction pattern as illustrated in Figure 41. A Semantic Catalogue supports the ability to improve the search for resources by by exploiting the conceptual relations between concepts defined in an ontology. The ontology should represent (a subset of) the thematic domain of the user.

On the front-end to a client application, the Semantic Catalogue provides an interface in form of the ORCHESTRA Catalogue Service and, on the back-end, it offers access to more than one catalogue service instances, possibly with different associated meta-information models, e.g. OGC Catalogue Services in the form of the ebRIM and ISO application profiles, any other non-OGC compliant catalogue service (e.g. UDDI in a Web service environment) or even an Intenet search engine. However, this structural diversity should be transparent to the user of the Semantic Catalogue component by means of cascaded catalogue OSIs.

First, a query to the Semantic Catalogue is analysed in a semantic query processor that uses the Ontology Access Service to expand the query according to related concepts in an ontology. It then generates individual queries and directs them to the appropriate meta-information sources. The response (result sets in the catalogue’s own structure) is then assembled and structured by a semantic report generator and returned as query response to the client.

Furthermore, there is the option to use the Annotation Service (see section 9.8.3) in order to annotate selected textual results against the ontology that has been used in the query expansion in order to assess and interprete the results in the context of the thematic domain.

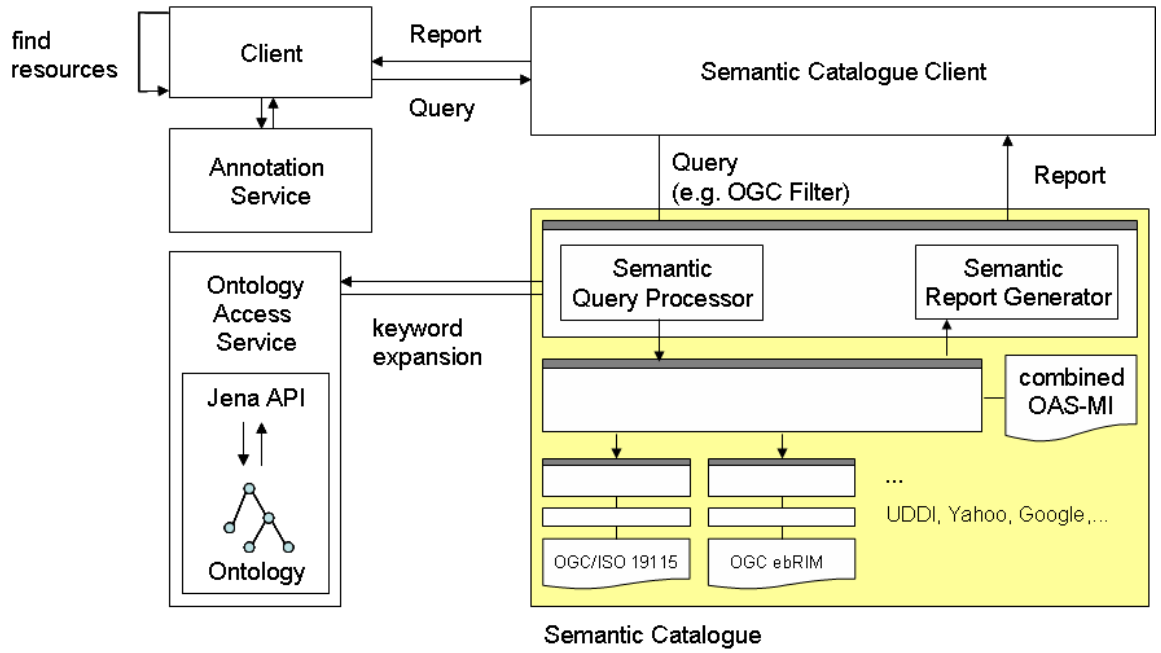

Figure 41: Example of a Semantic Catalogue

9.10.7 Naming in Dynamic OSN Environments

In the following, a usage of the Name Service (see section 9.7.6) in the case of dynamic OSN environments is described. Dynamic OSN environments are characterised by the fact that the assignment of OSIs to one or more OSNs may change during the lifetime of an OSI (e.g. due to a central OSN administrative decision or due to an autonomous decision of an OSI).

Note: Version 3 of the RM-OA will investigate how an OSI knows about the used naming policy for its own name and its (current) membership in an OSN.

In order to support a dynamic OSN environment, an interaction of Name Service instances is required. Consider the following cases:

- An OSI is added to OSN A and is not already registered at any Name Service instance. In this case, the OSI can be registered at the Name Service instance of OSN A. The Name Service creates a globally unique name for the OSI and can then be used to resolve the name.
- One or more OSIs are added to OSN A and these OSIs are already registered at a Name Service instance of OSN B. As these OSIs already have names, the Name Service instance of OSN A is not used to create OSI names. Instead, a mechanism is needed to create a linkage between the Name Services instances of OSN A and OSN B. Such a mechanism is further described below.
- An OSI is removed from an OSN. If the OSI is not member of another OSN, it may be deregistered from the Name Service instance of the OSN, which means that it will lose its name. However, it may also be useful to keep its name and registration in order to use the OSI in another OSN.
- A new OSN is created and OSIs are added as described above. The new OSN may establish a new Name Service instance or may reuse an existing one of another OSN.
- An OSN is removed which implies that all its OSIs are removed from that OSN. The Name Service instance of the OSN may still be used by another OSN.

The following figure illustrates a linkage between two Name Service instances.

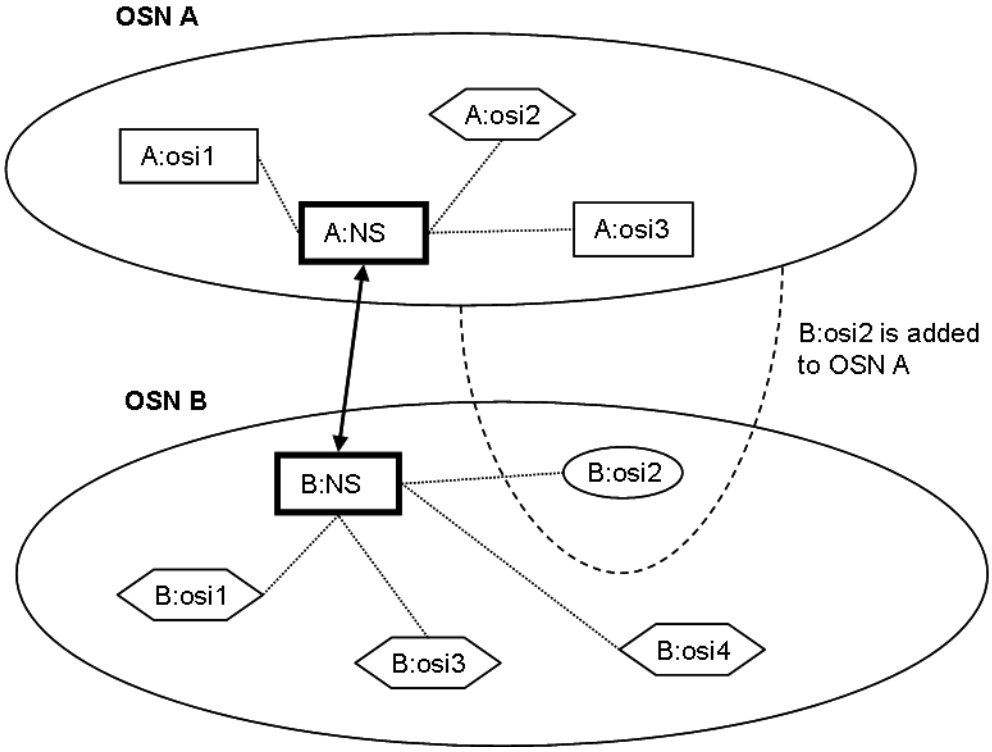

Figure 42: Linkage between Name Services

The figure shows two OSNs which are initially separated. Each OSN has its own Name Service instance indicated by A:NS and B:NS. Each OSI is registered at the Name Service instance of its OSN, indicated by the connecting lines. Now, B:osi2 from OSN B is in addition added to OSN A.

As the added OSI is registered at B:NS, a linkage is established between A:NS and B:NS. The linkage is used for name resolution in the following way. In order to resolve a name within OSN A, A:NS is used. If A:NS is not able to resolve the name among its registered OSIs, it uses the linkage and directs the request to B:NS. Thus, cascading name resolution is performed. This allows the resolution of the name of B:osi2 using A:NS.

Note that the linkage may be used in both directions for cascading name resolution. B:osi2 may use its original Name Service B:NS to resolve names within OSN A and OSN B.

A:NS and B:NS may use different naming policies.

To support linkage of Name Service instances, the Name Service has an additional interface called *NameServiceLinkage* that includes the following operations:

linkNameService(PSI)

This operation establishes a linkage between this Name Service instance and another one which is specified by its PSI within the current service platform. The linkage is used to allow for cascading name resolving. This means, if this Name Service instance has no information to map an OSI name to a PSI or vice versa, it can redirect the request to all linked Name Service instances.

unlinkNameService(PSI)

This operation removes a linkage between this Name Service instance and another one which is specified by its PSI within the current service platform.

10 Technology Viewpoint

According to the ORCHESTRA Reference Model as introduced in section 5.3, the Technology Viewpoint specifies the technological choices of the service platform and its operational issues. Thus, when considering the design process of the ORCHESTRA Service Network, it contains the specification of the service platform and its characteristics upon which the ORCHESTRA Services and ORCHESTRA Application Schemas are to be mapped.

The present RM-OA document, being a reference model for the design of ORCHESTRA Service Network, only contains the guidelines and requirements for the platform specification. It comprises the following parts:

- a specification of all properties that are required to be compliant with the SOA Reference Model of OASIS,
- a specification of how the UAA mechanisms are intrinsically supported by the platform,
- agreement on the usage of specific data formats (e.g. non-GML representation of coverages),
- a specification of a bijective mapping of the platform-specific schema language from and to UML (both for information models and for service types) according to the OMM,
- a specification of possible restrictions of the platform, e.g. to be considered in the service mapping process.

10.1 Specification of Platform Properties

Being a realisation of the OMM meta-class OMM_PlatformSpec (see section 9.2.2.2), a platform specification has to define the following set of properties, which are considered in the context of the OASIS Reference Model for Service Oriented Architecture 1.0 (SOA-RM, 2006). As an important example see the platform specification as part of (ORCH-ImplServ 2007) for the “ORCHESTRA Web Services Platform”.

1. RM-OA Version

   Version number of the RM-OA to which the platform specification refers to.

2. Platform Name

   Name of the platform. In case of a standard platform, a reference shall be provided.

   Example: “ORCHESTRA Web Services Platform”

3. Interface Language

   Specification of the formal language used to define SOA-RM Service Interfaces. In case of a standard language, a reference shall be provided.

   Example: Web Services Description Language (WSDL)

4. Execution Context

   Specification of the SOA-RM Execution Context. The Execution context is an agreement between service providers and consumers. It contains information that can include preferred protocols, semantics, policies and other conditions and assumptions that describe how a service can and may be used. This includes, for example, the specification of the transport and the security layer, the format of the messages exchanged between service providers and consumers, etc. In case of a standard SOA-RM Execution Context, a reference shall be provided.

   Example: SOAP 1.2 HTTP binding for message transport, WS-Security in conjunction with SLL shall be used if encryption of messages is required, etc.

5. Schema Language

   Specification of the schema language used to define SOA-RM Information Models. The schema language defines the platform dependent encoding of a platform independent information

model as specified by an ORCHESTRA Application Schema.

Example: XML-Schema and a GML Profile based on the GML Simple Feature Profile.

6. Schema Mapping

   Specification of how to map the abstract level (UML) to the schema language used for this particular platform.

   Example: XML-Schema/GML encoding rules for ORCHESTRA Application Schemas (ISO 19136 Annex E and F + additional rules for non-GML types)

7. Information Model Constraints

   Specification of the constraints on the SOA-RM Information Model, especially the constraints on the message format required to accomplish the SOA-RM Action model.

Furthermore, in the following sections some specific aspects are discussed that have to be considered on a platform level in order to increase the level of interoperability.

10.2 Selection of User Management, Authentication and Authorisation Mechanisms

The RM-OA concept for User Management, Authentication and Authorisation (UAA) and the respective abstract specifications are by intention specified at a high level of abstraction in order to be able to cope with established UAA mechanisms for dedicated platforms. Thus, a platform specification has to define how the ORCHESTRA UAA concept can be realised for a specific platform. This includes an agreement on the authentication and authorisation mechanisms permitted within an OSN, the transport and handling of session information among OSCs, the selection of a language for the expression of permissions and possibly the predefinition of common permissions and default subjects and principals. Some aspects of these definitions, especially the technical details, may not necessarily be part of the platform specifications but of the implementation specifications of the UAA services.

Example: The Authentication Service implements a simple username/password mechanism, and the Authorisation Service a role-based access control (RBAC) system. Additional authentication and authorisation mechanisms are not supported. Session information will be exchanged by means of a platform specific protocol, for example inside the header of a SOAP message.

10.3 Agreement on Data Formats

A platform specification may also contain an agreement on the usage of (de-facto or de-jure) standard data formats (e.g. MIME types) and specific, often proprietary data formats to be exchanged between OSCs.

Example: An agreement on well-known coverage representation formats (e.g. GeoTIFF, HDF) to represent coverage type information which is not encoded in GML.

10.4 Definition of a Reversible Platform Mapping for Information Models

Since an information model may also be modelled directly in a platform-specific schema language without the need to follow the OMM approach of defining an OAS and applying platform specific mapping rules, the conformance of such information models to the OMM has to be ensured.

It must be possible to generate the UML representation of a given information model, modelled in a platform specific schema language, to check compliance to the OMM. Therefore the definition of encoding rules for the mapping of an OAS to a platform specific transfer format must not be ambiguous and has to be specified as a reversible mapping as part of the platform specification.

A platform specification may also include an optional annex providing procedures and guidelines for how these mapping rules shall be applied.

Examples:

1) Usage of the reversible encoding rules from ISO 19136 Annex E and F for the platform "Web Services" to map (ORCHESTRA) Application Schemas to GML.
2) Provision of a table that maps basic UML data types (see section 8.7.2.2) to basic XML-Schema

data types and vice versa (e.g. CharacterString ⇔ xsd:string). See also ISO 19136 Annex D.

3) Guidelines for the usage of UML to GML Application Schema tools.

10.5 Definition of Procedures for the Mapping of Service Interfaces

Procedures for the mapping of the platform-neutral service interfaces to a specific interface language may have to be defined. These procedures shall ensure that the mapping is in compliance with the rules of the ORCHESTRA Service Meta-Model (OMM-Service, see section 9.2). The procedures should be defined in an optional annex of the platform specification. The mapping itself shall be part of an implementation specification. If this can be accomplished, such a mapping should be bi-directional and described in a machine readable way.

Example: Description of how to transform XMI to WSDL using Enterprise Architect.

Note 1: In cases where ORCHESTRA Services are directly specified on a platform level, compliance with the OMM-Service must be assured for interoperability reasons.

Note 2: When mapping interfaces and exceptions to a service platform (e.g. Web Services), the relevant standards have to be considered, such as e.g. WS-I Basic Profile V1 (http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html). It refers to a set of non-proprietary Web services specifications, like SOAP, WSDL etc., along with clarifications to and amplifications of those specifications in order to promote interoperability.

10.6 Restrictions on certain Services

A platform specification may further reduce the complexity or restrict the scope of certain services, if this is required to meet the main characteristics of the selected platform.

Note that this complicates interoperability between different platforms. There should exist a bi-directional mapping between an abstract and an implementation specification and this mapping should be described in a machine readable way.

Example: A platform “OGC Web Services” may permit the mapping of some OA Services to OGC service interfaces by knowingly allowing a derivation from the abstract service interface specifications.

11 Engineering Viewpoint

According to the ORCHESTRA Reference Model as introduced in section 5.3, the Engineering Viewpoint specifies the mapping of the ORCHESTRA service specifications and information models to the chosen service platform and the specification of the characteristics of ORCHESTRA Service Networks.

Thus, when considering the design process of the ORCHESTRA Service Network, the mapping process itself belongs to the Engineering Viewpoint. It is documented in corresponding sections of the implementation specifications, see (ORCH-ImplServ 2007).

The present RM-OA document, being a reference model for the design of ORCHESTRA Service Network, restricts the description of the Engineering Viewpoint to the discussion on OSN Characteristics.

Note: The following sections in the RM-OA Engineering Viewpoint are preliminary ideas and need to be validated and formalised during the course of the ORCHESTRA project when further implementation experiences have been gained. Results of this validation will go into version 3 of the RM-OA.

11.1 OSN Characteristics

11.1.1 Policies

An ORCHESTRA Service Network (OSN) is defined as a set of networked hardware components and ORCHESTRA Service Instances (OSIs) that interact according to defined policies in order to serve the objectives of ORCHESTRA Applications (see section 5.3.3). Thus, the basic units within an OSN for the provision of functions are the OSIs, whereas their interaction principles are determined and characterized by policy definitions. Instead of pre-determining a specific policy for all possible OSNs, the following sections of the RM-OA only define policy elements and rules for the definition and the existence of policies for different OSN characteristics. Using this approach, the policies of an OSN may be set-up according to the given individual business and organisational needs and models.

Note that this approach does not fix the model for policy enforcement, be it centralised or decentralised. Furthermore, it does not prescribe the time and the way that the policies are defined, be it (pre-) determined by a central authority or negotiated online between the participating parties. Thus, a wide spectrum may be covered, from a centrally-administered OSN with a high level of access control and a fixed and pre-defined list of OSIs up to an open and flexible OSN with dynamic registration and de-registration of OSIs and a distributed administration.

An OSN is characterized by adopting a harmonised approach for the following policies (non exhaustive list):

- resource identification
- resource discovery
- OSN operation
- UAA (User Management, Authorisation, Authentication)

An OSN is defined by a specification of at least these policies. This OSN specification should be accessible in one of the following forms:

- in written form by the institution running the OSN
- by querying the OSN catalogue in a centralised discovery policy (see section 11.1.3) or
- by querying one of the catalogues in a decentralised discovery policy (see section 11.1.3).

11.1.2 Resource Naming Policy

The Resource Naming policy of an OSN deals with the question of how resources in OSN-like service instances and feature instances are identified. The uniqueness of resource identifiers in an OSN and across OSNs is further discussed in the section 11.3.

The Resource Naming policy is defined by the following elements:

- name service: statement if a Name Service (see section 9.7.6) is used that is responsible for the provision of globally unique identifiers for OSIs and/or feature instances.
- naming policy for service instances: specification of which naming policy is used for the identification of OSIs across platforms. Currently, the following approach has been identified (see section 11.3.1):
  - platform as namespace: The global uniqueness of OSIs is enforced by using the service platform as namespace, i.e. the platform-specific identifier of an OSI is used.
- naming policy for feature instances: specification of which naming policy is used for the identification of feature instances. Currently, the following approach has been identified (see section 11.3.2):
  - OSI as namespace: Each OSI that acts in the role of a Feature Access Service shall be responsible for managing a namespace of related feature instance identifiers.

11.1.3 Resource Discovery Policy

The Resource Discovery policy of an OSN deals with the registration of resources in an OSN. Registration means the creation of an associated meta-information entry for a resource in a catalogue in order that a user who is part of the information community of that catalogue may discover the resource (see section 0).

The process of registration as well as the process of discovery is supported by operations specified in the Catalogue Service (see section 9.7.5). A resource may be registered in one or more catalogues.

The meta-information about resources is defined in OAS-MI according to the rules of the OMM Information Model. A resource may be the OSN itself, feature types and instances, service types and instances and UAA resources such as subjects (see section 7.5.2).

The Resource Discovery Policy is defined by the following elements:

- discovery policy: statement about the discovery policy used in the OSN. Possible alternatives are:
  - centralised discovery: There is a distinguished Catalogue OSI (called OSN Catalogue) that serves as the “entry point“ to the OSN.

    Note: The presence of an OSN Catalogue does not exclude the existence of other instances of the Catalogue Service.
  - decentralised discovery: All instances of the Catalogue Service are equivalent.
  - no discovery: There is no Catalogue OSI. This means that the service interactions are not mediated through an instance of a Catalogue Service.

    Note: Whether a network of OSIs without a discovery capability based on a Catalogue OSI is a “valid” OSN is under discussion.
- definition of the catalogues used in the OSN
  - name of the OSN Catalogue
  - statement if the catalogue is the “OSN Catalogue”, i.e. the “entry point” of an OSN (see above).
  - query language of the OSN Catalogue
  - ORCHESTRA Application Schema for Meta-information (OAS-MI) of the OSN Catalogue for the purpose of discovery
  - resource types that may be discovered through the OSN Catalogue
    - OSN
    - feature types

        - feature instances
        - service types
        - service instances
        - subjects
        - … others
    - ORCHESTRA Application Schema for Meta-information (OAS-MI) of the OSN Catalogue for the purpose of service invocation, i.e. the OAS-MI for the default service capabilities for all OSIs running in the OSN.

      Note: The default service capabilities usually correspond to the OAS-MI for service instance discovery (see above). However, this is not obligatory.

11.1.4 OSN Operating Policy

The OSN Management Policy is divided into three sub-policies which are described in the following sections:

- OSN management policy
- service management policy
- network management policy

11.1.4.1 OSN Management Policy

The OSN Management Policy deals with the requirements concerning the management and the operation of an OSN. It is defined by the following elements:

- general administrative information
    - name: globally unique name of an OSN.

      Note: An example for such a name is the name of the OSN Catalogue (see section 11.1.3) if the name of the OSN Catalogue is globally unique.
    - description: human-readable textual information about the goals and purpose of the OSN.
    - OSN provider: Information about the community, institution or organisational unit operating the OSN in the sense of defining and enforcing the policies of the OSN.
    - administrators: Names and addresses of those persons who are responsible for the operation of the OSN.
- Technical Information
    - platform: reference to the platform specification upon which the OSN is based

      Note: Currently, an OSN may only run on top of one specified platform.
    - name and platform-specific identifier (OSI) of the “OSN Catalogue” (if any, see section 11.1.3) as the entry point to the OSN
    - requirements for all OSIs interacting in the OSN:
        - minimal required set of formats (see the *acceptFormats* parameter of the *ServiceCapabilities* interface as specified in section 9.6.1) that every OSI has to support for the *getCapabilities*-operation
        - minimum required level of security to be provided by each OSN component (client or OSI). The security policy shall make statements (e.g. technologies or platform-specific mechanisms used) about the following topics:
            - encryption of communication

Note: There are some limitations by law in some countries about the usage of encryption and some sort of communication technology (e.g. France).

- measures against intrusion, alteration, eavesdropping, non-repudiation

- applicable classifiers of the OSN (see section 11.2)
- service registration: statement about whether a service can be registered at any time by any subject (open service registration) or whether the service registration is controlled (controlled service registration based on a resource discovery policy, see section 11.1.3).
- list of mandatory services within the OSN, i.e. at least one OSI of this service type shall be operational in an OSN. This list may be derived from the type of OSN or listed explicitly.
- list of additional services for whoch OSIs may be registered in at least one of the catalogues are allowed. The alternatives are:
  - any OSI of any service type is allowed
  - no OSIs other than the mandatory services are allowed
  - a specified number of OSIs of a specified list of service types are allowed

11.1.4.2 Service Management Policy

The Service Management Policy deals with the administrative requirements that OSIs of a specific service type have to fulfil when interacting within a specific OSN. It is defined by the following elements:

- service monitoring
  - list of service and network events to be monitored (e.g. just make calls to *getCapabilities*())
  - list of OSIs to be monitored (e.g. all OSIs that are registered in the OSN catalogue) and supported statistics about the usage of services in an OSN (see Service Monitoring Service described in section 9.7.10)
  - list of conditions under which management notifications have to be generated
- quality of service (both on the level of individual or all OSIs of a given service type or generally for all OSIs deployed in an OSN)
  - availability of service (e.g. work hours, 24x7, redundant)
  - maximum response time for service operations

11.1.4.3 Network Management Policy

The Network Management Policy deals with the management of the communication resources of the specified platform. For this part of the OSN Operating Policy, the RM-OA refers to the corresponding management standards that are specific for the chosen platform and underlying communication protocols, e.g.

- for protocols based on the Internet protocol stack (IETF RFC 1122 Requirements for Internet Hosts -- Communication Layers), these are the IETF recommendations related to RFC 2570 Introduction to Version 3 of the Internet-standard Network Management Framework.
- for protocols based on ISO/OSI 7498-1 Open Systems Interconnection, these are the ISO standards related to ISO/OSI 7498-4 Management Framework.

11.1.5 User Management, Authorisation and Authentication Policy

The User Management, Authorisation and Authentication policy of an OSN is divided into the following sub-policies:

- user management policy
- authentication policy
- authorisation policy

There are many different concepts and technologies in the context of user management, authorisation and authentication. Often, these concepts and technologies cannot be applied independently from each other. Thus, it must be ensured that the policies are specified coherently.

11.1.5.1 User Management Policy

The User Management Policy deals with the way users are represented and made known (registered) in an OSN. It is defined by the following elements:

- subject information: minimum information to be provided when specifying a subject. This corresponds to a dedicated meta.information schema for the purpose of “user profiling” (see section 8.4.2.5).
- dynamic registration of users: statement about whether dynamic registration (i.e. registration and de-registration of users at runtime) is allowed or not. In case it is allowed the business process for dynamic registration shall be described for each of the following:
  - subjects (users, including ORCHESTRA Service Instances (OSIs)),
  - groups (group of subjects)

  A business process to register a new subject shall clarify responsibilities so that the liability for the registration of a new subject is explicitly expressed.
- pre-defined subjects and groups: statement about whether the OSN requires the existence of specific pre-defined subjects and groups.

11.1.5.2 Authentication Policy

The Authentication Policy deals with the generation of session information. It is defined by the following elements:

- set of allowed authentication mechanisms
  - default authentication mechanism
  - restrictions on the set of allowed authentication mechanisms
- representation of principals: specification of how principals are represented in an OSN (optional)

  Note: Even though the set of allowed authentication mechanisms determines the possible presentations of principals. It may be required for clarity to explicitly specify the representations of principals.
- single-sign-on or multiple authentication: statement whether single-sign-on and/or multiple authentication is used.
- treatment of session information: definition how session information is treated, either by a session key or by a session envelope
- session key validity: validity space for a “session key” returned by the Authentication Service after a successful authentication has to be assured

11.1.5.3 Authorisation Policy

The Authorisation Policy deals with the way the access to resources in an OSN is controlled. It is defined by the following elements:

- set of allowed authorisation paradigms (e.g. role based access control, trust management)
  - default authorisation paradigm for the whole OSN, i.e. for all OSIs of the OSN

  - authorisation paradigms that shall be applied for OSIs of a given service type or for individual OSI

- default permissions for pre-defined subjects and groups
- policy enforcement: statement about whether the authorisation takes place on the service level and/or the data level.

11.2 OSN Classifiers

In order to characterise OSNs and to provide constraints upon them for their classification, the policies described above are structured into policy elements. Depending on the scope of OSN that is to be designed a specification of these policy elements is either mandatory or optional.

A preliminary list of classifiers that may be attached to OSNs is given in Table 40. The main ideas are as follows:

- All OSNs shall use “platform as namespace” for the naming policy of OSIs and “OSI as namespace” for the naming policy of feature instances. These two policy elements are explained in section 11.3.

OSN Classifiers	Resource Naming	Resource Discovery	OSN Operating	UAA
Primitive	Platform as namespace for OSIs, OSIs as namespace for feature instances			
Mediated	dito	OSN Catalogue		
Managed	dito		Service Monitoring	
Access-controlled	dito			Harmonised UAA approach

Table 40: Minimum Policy Requirements according to OSN Classifiers

- For a “primitive OSN” there are no further constraints or rules, i.e. it may consist of an arbitrary network of OSIs as long as these OSIs have been designed according to the rules of the OMM. An example of a primitive OSN is a Web Mapping application for environmental information run by one institution that renders a fixed set of map layers based on a set of pre-defined OSIs whereby changes and extensions in this application are rare or even not expected.

  Note: As primitive OSNs do not necessarily support means for resource discovery, they do not, as a whole, comply with the architectural requirement of “self-describing components”, see section 6.3.7. However, for ORCHESTRA Applications with low requirements on flexibility this may be a reasonable solution. Nevertheless, as ORCHESTRA aims at supporting environments that are “designed for change”, OSNs that are only classified as “primitive” are usually not sufficient.

- A “mediated OSN” requires the usage of at least one catalogue OSI (called OSN catalogue) for the discovery of service and feature instances. An example of a mediated OSN is a Sensor Web application within or between institutions that shall be flexible enough in order to integrate a varying set of data source systems (e.g. databases with sensor-related information).and models.
- A “managed OSN” shall, out of the set of the policy elements for OSN Operation, at least support the policy element of “service monitoring” (see section 11.1.4.2). This means that it shall be possible to monitor the execution characteristics of OSIs. An example is a geo-statistical service that is offered to customers in a commercial environment such that the service provider needs to observe the availability of the running OSIs.

- An “access-controlled OSN” shall support a harmonised approach for the UAA policy elements as described in section 11.1.5. Examples are all applications that need to restrict the access to services and/or data to end-users with a well-defined role (e.g. a risk manager).
- Of course, several classifiers may be applicable to one OSN, e.g. there may be an OSN that is both mediated, managed and access-controlled. Such OSNs are capable of exploiting the full potential of the ORCHESTRA approach and respond to the architectural requirements stated in section 6.3..

11.3 Naming Policy Examples

11.3.1 Platform as Namespace for OSIs

In the following a naming policy approach for OSIs is presented wherein the assignment of a name to an OSI is independent of the membership of an OSI in an OSN. In particular, a unique OSN name and an OSN-related namespace are not required for this approach.

According to the ORCHESTRA Architecture, an OSN is designed to be based on one or several service platforms. A service platform provides the basic communication and encoding mechanisms for the service interactions (the service infrastructure). By definition, an OSI is the result of a platform-specific deployment step making the OSI part of a certain platform domain. Thus, an OSI can be considered a service in the sense of the used service platform.

One of the characteristics of a service platform is that a service is identified by means of a platform-specific service identifier which is unique within the platform. The identifier is usually assigned when the service is deployed, i.e. entered into the platform. The service platform acts as a namespace for OSIs.

Service Platform Examples:

- Platform W3C Web Services: An OSI corresponds to a Web Service according to the W3C specifications. A Web Service is identified by a URI. A URI is a globally unique identifier for all Web Services.
- Platform Java RMI: An OSI corresponds to a Java Object which is remotely accessible and published in an RMI registry. The Java Object is identified by a URI (with an empty schema), i.e. a string of the form

  //<host>:<port>/<name>

  where <host> and <port> are used to locate the registry. <host> is a hostname (IP-Address or domain names according to DNS) and <port> is the host-specific port number. <name> is the published name of the Java Object which is unique within the registry.
- Platform CORBA: An OSI corresponds to a CORBA Server Object. In CORBA objects can be uniquely identified by an IOR (Interoperable Object Reference). Another way is to use the address of a Name Service and a name local to the Name Service in a similar way as for the Java RMI example.

In the current RM-OA version, it is assumed that a given OSN is based on just one pre-selected service platform. Thus, the service identifier of that platform can directly be used to name the OSIs. As the service identifier is unique within the platform and only one platform is used, the resulting OSI names are unique.

RM-OA version 4 will consider an enhancement of this naming policy for the case that an OSN spans several platforms.

11.3.2 Feature Access OSI as Namespace for Feature Instances

In the following a naming policy approach for feature instances is presented wherein the assignment of a feature instance identifier is combined with the identifier of the Feature Access OSI that provides access to the feature instance.

Thus, a feature access OSI manages a namespace of feature identifiers. The feature identifiers provided by such an OSI are initially not unique within the whole OSN, but only unique among all features

of that OSI. In general there may be multiple Feature Access OSIs in an OSN. In order to obtain unique identifiers, the name of the corresponding feature access OSI is added in order to get a unique identifier within an OSN.

Figure 43 provides an example: Three feature access OSIs are backed by different source systems. OSI a and OSI b are part of one OSN, OSI b and OSI c are part of another OSN. All feature instances related to these OSIs have identifiers f1, f2, f3 which are unique for each OSI. By adding the OSI names, the resulting feature identifiers a:f1, b:f1, c:f1 etc. become unique within the OSNs. They are even globally unique because the OSI names are globally unique.

This naming policy for the identification of feature instances is summarised as follows:

Within an OSN, each OSI that acts in the role of a feature access service shall be responsible for managing a namespace of related feature instance identifiers. Each such OSI shall assign identifiers to feature instances which are accessed using that OSI. Such an identifier shall be combined of two elements:

- the OSI name
- an OSI-specific identifier which unambiguously identifies the feature instance among all other feature instances of that OSI.

Together these elements form a feature identifier which is unique within the OSN.

Note: The naming policy just described only ensures the uniqueness of feature instances in an OSN regardless of their real-world phenomenon that they are representing. The situation in which two feature instances provide a (possibly) different view upon the same real-world phenomenon (e.g. a road) is a question of semantic identity that is to be solved on the semantic level of the information model framework (see Figure 20), possibly based on inferencing about an ontology and/or a knowledge base of the respective thematic domain.

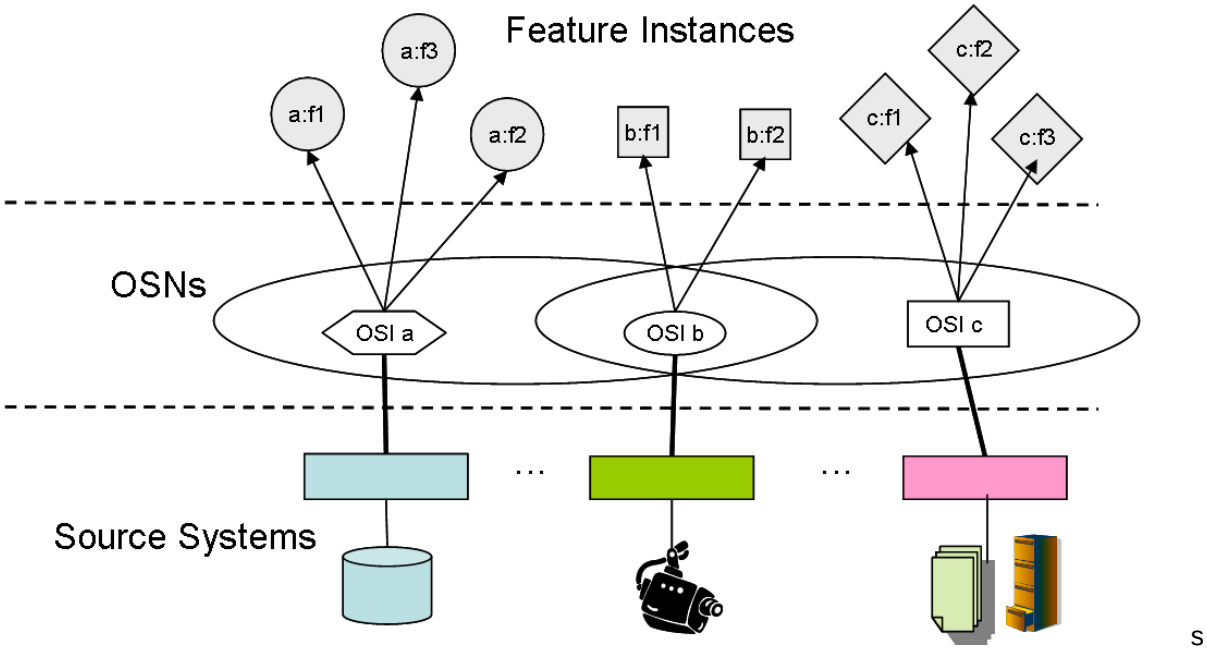

Figure 43: Constructing feature identifiers by using OSI-related namespaces

Constructing feature identifiers according to this rule has the following consequences:

- As each OSI name is globally unique as described in the previous section, the feature identifier is also globally unique.
- If the *createFeature* operation of the *FeatureAccessService* interface (see section 9.7.1) is used to create a new feature instance, the respective Feature Access OSI must assign a unique (i.e.

not yet used) feature identifier to it.

- The feature identifier can act as a locator for the feature. The OSI used to access that feature can be obtained from the feature identifier. A client requesting attributes of the feature can therefore direct its request to that OSI. In the same way as a uniform resource locator is used in the Web to locate a resource, a feature identifier is used to locate a feature instance within one or multiple OSNs.

The way a feature access OSI assigns identifiers to its feature instances is not further specified. In order to simplify the mapping between feature identifiers and the underlying feature information, certain feature type-dependent key attribute values may be used when constructing an identifier. However, this is very much source system dependent.

A feature access OSI may also support version management of features, i.e. it may allow access to various former versions of a certain feature instance. The current version and former versions may exist at the same time. In principle the current version and each former version of a feature instance can be considered separate instances which are implicitly or explicitly associated with each other. All these instances can be distinguished by their identifiers. The way versioning is reflected in the identifiers is not specified here.

Note: The principle of constructing a global identifier by combining an OSI name with an identifier which is unique within the context of that OSI can be used for identifying purposes wherever a globally unique identifier is needed.

12 Conclusion

The present RM-OA Revision 2.1 represents the understanding of the ORCHESTRA consortium about an open, generic and standards-based service-oriented architecture for distributed environmental and risk management applications after the third year of the project's runtime. Its focus is currently on syntactic interoperability whereby the upgrade towards the support of semantic interoperability has been prepared.

The following sections provide a summary of the major deviations of the RM-OA Design Decisions from ISO and OGC standards (section 12.1) and a short summary of the items that are intended to be covered in the outstanding version 3 of the RM-OA.

12.1 Summary of Deviations from Standards

Note 1: Textual changes are underlined.

Note 2: Deviations on the level of service types and abstract interface specifications are not listed here as most of the OGC and ISO service specifications are not provided on abstract level.

12.1.1 RM-ODP Computational Viewpoint mapped to RM-OA Service Viewpoint

In order to highlight the fact that an ORCHESTRA deployment will have the nature of a loosely-coupled distributed system based on networked services rather than a distributed application based on computational objects, the "computational viewpoint" will be referred to as "service viewpoint" in ORCHESTRA.

Rationale: section 5.2.2.

12.1.2 The OpenGIS Service Architecture (ISO 19119:2005)

In the ORCHESTRA Reference Model the distributed computing platform is referred to as the service infrastructure. However, the distinction between IT and GI services of ISO 19119:2005 is not applied for the ORCHESTRA service taxonomy because the ORCHESTRA Architecture (and thus the ORCHESTRA services) shall contain an integrated information model that covers thematic, temporal and spatial aspects.

Rationale: section 5.4

12.1.3 ISO 19101 Service Taxonomy

Workflow/Task services are services for support of specific tasks or work-related activities conducted by humans or software components with a high degree of autonomy (agents). These services support use of resources and development of products involving a sequence of activities or steps that may be conducted by different persons.

Processing services are services that perform computations. These computation might range from the performance of mathematical equations up to large-scale computations involving substantial amounts of data.

Rationale: section 5.4.2

12.1.4 ISO 19119:2005 Requirements for Platform-Neutrality

As part of the engineering viewpoint, the ORCHESTRA platform-neutral models are mapped to a specific service infrastructure context. The resulting platform-specific service models may be defined in UML or in terms of the platform-specific language (e.g. WSDL). However, it is required that a description of their mapping to the corresponding platform-neutral models be maintained. This mapping shall show how the intentions of the platform-neutral specifications are met in the context of the service platform. In order to support interoperability, the reverse mapping back to the concepts in the platform-neutral model must be defined (instead of should be defined).

Rationale: section 5.4.1

12.1.5 ORCHESTRA as Simple Service Architecture according to ISO 19119:2005

- Known service type

All ORCHESTRA service instances are of specific service types and the client may access the service type description prior to calling the service. In the ORCHESTRA Reference Model, a "known service type" is a service type with an externally available description.

Rationale: section 5.4.3

Note: The RM-OA version 3 will contain a more refined assessment if the ORCHESTRA Architecture may be considered as a "Simple Service Architecture" in the sense of ISO 19119 taking into account the latest developments about UAA and service chaining in the ORCHESTRA project.

12.1.6 The ORCHESTRA Definition of a Feature

One basic concept of the RM-OA Information Viewpoint is the feature, where a feature is an abstraction of a real world phenomenon perceived in the context of an ORCHESTRA Application. The ORCHESTRA definition of features explicitly goes beyond geographic features. It includes tangible objects of the real world but also abstractions, concepts or software artifacts (e.g. documents, software components of IT systems) that may have a representation only in software systems. These features may, but need not, have spatial characteristics. The ORCHESTRA understanding of a "real world" explicitly includes these hypothetical worlds or worlds of human thoughts.

Rationale: section 8.2

12.1.7 The ORCHESTRA Meta-Model (OMM)

The OMM is derived from the basic ideas of the ISO 19109 GFM, but it is not a true profile of it. The OMM is an evolution of the ISO 19109 GFM, taking into account additional, ORCHESTRA-specific requirements. In particular:

- The OMM extends the GFM by aspects of services modelling (see the OMM Service Metamodel (OMM-Service) in section 9.2).
- The OMM does not mandate the usage of one particular meta-information model (e.g. ISO 19115) as prescribed by the GFM. Instead, it gives the OSN designer the freedom to specify the meta-information models as required for the various purposes. It only mandates that an application schema for meta-information (OAS-MI) be specified according to the rules of the OMM-Information (see section 8.8).

Rationale: section 8.7

12.2 Evolution of the RM-OA

It is envisaged to mainly tackle semantic extensions of the OA in future versions of the RM-OA. This may encompass the following aspects (this is a non-binding and non-exhaustive list):

- Extension of the OMM-Service by including aspects of Semantic Web Services, e.g. semantic description of services (e.g. based on WSML, OWL-S or WSDL-S) as part of their meta-information, usage of semantics in advanced versions of ORCHESTRA Service Types (concerned sections: 9.2).
- Usage and influence of ontologies for the RM-OA Information and Service Viewpoints (concerned sections: 8.7, 9.2), e.g. comparision of the OMM and the OA services with the WSMO and the OWL-S framework.
- Support of further cases (e.g. service mediation) in the service mapping specification in addition to the service profile (concerned section: 9.2.9), e.g. discussion how the WSMO concept of mediation could be re-used for this question.