

Open Geospatial Consortium Inc.

Date: 2006-02-10

Reference number of this OGC® Project Document: **OGC 05-111r2**

Version: 1.0.0

Category: OpenGIS® Discussion Paper

Editor: *Roland M. Wagner*

OWS-3 GeoDRM Thread Activity and Interoperability Program Report: Access Control & Terms of Use (ToU) “Click-through” IPR Management

Copyright notice

Copyright © 2006 Open Geospatial Consortium, Inc. All Rights Reserved.
To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>

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: **OpenGIS® Discussion Paper**
Document subtype:
Document stage: Approved
Document language: English

Contents

I.	PREFACE.....	7
II.	SUBMITTING ORGANIZATIONS.....	7
III.	DOCUMENT CONTRIBUTOR CONTACT POINTS.....	8
IV.	REVISION HISTORY.....	8
V.	FUTURE WORK.....	8
	FOREWORD.....	9
	INTRODUCTION.....	11
1	RELATIONSHIP TO OTHER ACTIVITIES.....	12
2	USE CASES.....	13
2.1	USE CASE 1: ANONYMOUS USER.....	13
2.2	USE CASE 2: ANONYMOUS USER OF REMOTE SERVICE.....	13
2.3	USE CASE 3A: NAMED USER.....	14
2.4	USE CASE 3B: NAMED USER WITH PROOF.....	14
2.5	USE CASE 4: SERVICE CHAINING - OUT OF BAND NEGOTIATION.....	14
2.6	USE CASE 5: SERVICE CHAINING - IN-BAND TERMSOFUSE NEGOTIATION.....	15
2.7	USE CASE 6: SERVICE CHAINING - IN-BAND TERMSOFUSE (MULTIPLE CASCADING).....	16
3	THREAD REQUIREMENTS.....	17
3.1	INTEGRATION OF NEW FUNCTIONALITIES (E.G. TERMS NEGOTIATION) WITH EXISTING CONTENT FUNCTIONS.....	17
3.2	SESSION MANAGEMENT.....	17
3.3	STATE MANAGEMENT.....	17
3.4	ACCESS CONTROL.....	19
3.5	FULLY-INFORMED AND TRAIL & ERROR APPROACH.....	19
3.6	DIFFERENT HTTP TECHNOLOGIES: GET, POST AND SOAP.....	19
3.7	EXPLICIT AND IMPLICIT DESCRIPTION AND PROCESSES.....	19
3.8	BACKWARDS COMPATIBILITY.....	19
3.9	DIFFERENT PACKAGING OF NEW BUSINESS FUNCTIONALITY.....	20
3.9.1	<i>Stand-alone Variant</i>	20
3.9.2	<i>Fully integrated Variant</i>	20
4	IMPLEMENTATION: CUBEWERX/DSS/METALOGIC.....	21
4.1	DACS OVERVIEW.....	21
4.2	DACS ACCESS CONTROL SERVICE – THE ACS MODULE.....	22

4.2.1	<i>Module-to-ACS Protocol</i>	23
4.2.2	<i>Credentials</i>	25
4.3	DACS NOTICE ACKNOWLEDGEMENT.....	26
4.4	MIDDLEWARE SUPPORT	28
4.4.1	<i>Simple Mode</i>	28
4.4.2	<i>Secure Mode</i>	28
4.5	IMPLEMENTATION OF A GETUNSATISFIEDPRECONDITIONS SERVICE IN DACS..	29
4.5.1	<i>Testing Access</i>	30
4.5.2	<i>XML Output</i>	32
4.5.3	<i>Identity interoperability</i>	32
4.5.4	<i>dacs_auth_agent</i>	33
4.5.5	<i>Warning</i>	33
4.6	CUBEXPLOR-DACS-CUBESERV WORK FLOW	34
4.6.1	<i>Introduction</i>	34
4.6.2	<i>The start of the Workflow</i>	34
4.6.3	<i>Coarse-grained License Management</i>	34
4.6.4	<i>Fine-grained License Management</i>	36
4.6.5	<i>The actual getmap Request</i>	37
4.6.6	<i>Request Types other than getmap</i>	37
4.7	NOTICE ACKNOWLEDGEMENT TOKEN SPECIFICATION.....	37
4.7.1	<i>The Notice Acknowledgment Token</i>	38
4.7.2	<i>NAT Syntax</i>	38
4.7.3	<i>Encoding for Transport</i>	44
4.7.4	<i>Implementation Notes</i>	44
4.7.5	<i>See also</i>	46
4.7.6	<i>Author</i>	46
4.7.7	<i>Copying</i>	46
5	IMPLEMENTATION: UNIBW	47
5.1	GENERAL APPROACH	47
5.2	IMPLEMENTED USE-CASES	48
5.3	THE AGREEMENT WORKFLOW.....	48
5.4	CLICK-THROUGH LICENSING FOR NAMED USERS	49
5.4.1	<i>WS-Security and token profiles</i>	50
5.4.2	<i>Access Control for OGC SOAP messages</i>	50
5.5	CLICK-THROUGH LICENSING FOR NAMED USERS	53
5.5.1	<i>getSession Definition</i>	53
5.5.2	<i>Example (SOAP)</i>	53
5.5.3	<i>getSession definition in capabilities documents</i>	54
5.6	SERVICE EXCEPTIONS	55
5.7	DISCLAIMER NOT AGREED.....	55
5.8	INVALID CREDENTIALS	55
5.9	SERVICE CHAINING: FPS / CASCADING WMS	56
5.10	SOFTWARE IMPLEMENTATION DESCRIPTION	58
5.10.1	<i>Architecture</i>	58
5.10.2	<i>Components</i>	59

5.10.3	<i>Licensing of software</i>	60
5.10.4	<i>Tests and Demonstrations</i>	60
6	IMPLEMENTATION: LAT-LON	61
6.1	GETLICENSES OPERATION	61
6.1.1	<i>GetLicences request</i>	62
6.1.2	<i>GetLicences request example</i>	62
6.1.3	<i>GetLicences response</i>	62
6.1.4	<i>GetLicences response examples</i>	63
6.1.5	<i>Discussion</i>	63
6.2	NEGOTIATERMS OPERATION	65
6.2.1	<i>NegotiateTerms request</i>	65
6.2.2	<i>NegotiateTerms request example</i>	65
6.2.3	<i>NegotiateTerms response</i>	65
6.2.4	<i>NegotiateTerms response example</i>	65
6.2.5	<i>DRM WFS response without license acceptance</i>	66
6.2.6	<i>Discussion</i>	66
6.3	A CLIENT APPLICATION DEMO	67
6.3.1	<i>Click-through on a free feature type</i>	68
6.3.2	<i>Click-through on a non-free Feature Type</i>	70
6.4	CONSEQUENCES IN REGARD TO OGC SPECIFICATIONS	71
6.4.1	<i>OWS Common Change Request</i>	71
6.4.2	<i>GML Change Request</i>	71
7	GENERAL GEODRM CAPABILITIES AND TERM-OF-USE MODELS	72
7.1	REQUIREMENTS	72
7.2	SCHEMA OVERVIEW	73
7.2.1	<i>Main Axis</i>	73
7.2.2	<i>Matching: Product – Resources</i>	74
7.2.3	<i>Preconditions</i>	76
7.2.4	<i>Terms Management</i>	77
7.2.5	<i>Workflow</i>	77
7.3	SCHEMA DESIGN	78
7.3.1	<i>Element Authentication</i>	78
7.3.2	<i>Element GeoDRM Capabilities</i>	78
7.3.3	<i>Element GeoDRMPreConditions</i>	78
7.3.4	<i>Element GeoDRMPreConditions/TermsManagement</i>	78
7.3.5	<i>Element GetCapabilities</i>	79
7.3.6	<i>Element NegotiatePreConditionsRequest</i>	79
7.3.7	<i>Element NegotiatePreConditionsRequest/ProductCatalog</i>	79
7.3.8	<i>Element NegotiatePreConditionsRequest/ProductCatalog/Product</i>	79
7.3.9	<i>Element NegotiatePreConditionsResponse</i>	80
7.3.10	<i>Element OnlineResource</i>	80
7.3.11	<i>Element PricingAndOrdering</i>	80
7.3.12	<i>Element Product</i>	80
7.3.13	<i>Element ProductCatalog</i>	81
7.3.14	<i>Element ProductCatalog/Product</i>	81

7.3.15	<i>Element TermsManagement</i>	81
7.3.16	<i>Element WorkflowOfOperations</i>	81
7.3.17	<i>ComplexType CapabilitesType</i>	82
7.3.18	<i>ComplexType ProductSubsetType</i>	82
7.3.19	<i>Element ProductSubsetType/TAN</i>	82
7.3.20	<i>ComplexType ProductType</i>	83
7.3.21	<i>Element ProductType/GeoDRMPreConditions</i>	83
7.3.22	<i>Element ProductType/GeoDRMPreConditions/TermsManagement</i>	83
7.3.23	<i>Element ProductType/Resource</i>	83
7.3.24	<i>Element ProductType/Resource/ResourceRecord</i>	83
7.3.25	<i>Element ProductType/Resource/ResourceRecord/ResourceCapabilities</i> . 84	
7.3.26	<i>Element</i> <i>ProductType/Resource/...../ResourceCapabilities/EmbeddedCapabilities</i>	84
7.3.27	<i>Element ProductType/Resource/ResourceRecord/ResourceType</i>	84
7.3.28	<i>Element ProductType/Resource/ResourceRecord/ResourceId</i>	84
7.3.29	<i>Element ProductType/Product</i>	84
7.3.30	<i>Element ProductType/TAN</i>	85
7.3.31	<i>ComplexType TermsManagementSubSetType</i>	85
7.3.32	<i>Element TermsManagementSubSetType/terms</i>	85
7.3.33	<i>ComplexType TermsManagementType</i>	85
7.3.34	<i>Element TermsManagementType/terms</i>	85
7.3.35	<i>Element TermsManagementType/terms/EmbeddedTerms</i>	85
7.3.36	<i>ComplexType WorkflowOfOperationsType</i>	86
7.3.37	<i>Element WorkflowOfOperationsType/OperationName</i>	86
7.4	EMBEDDING METHODS	86
7.4.1	<i>Backwards compatible embedding</i>	86
7.4.2	<i>Current and upcoming specification embedding</i>	86
8	COMMON ELEMENTS	87
8.1	PROCESS MODEL.....	87
8.1.1	<i>Information phases</i>	88
8.1.2	<i>Negotiation phase</i>	89
8.1.3	<i>Contracting phase</i>	89
8.1.4	<i>Delivery phase</i>	90
8.2	INFORMATION MODEL	90
8.2.1	<i>Capabilities</i>	90
8.2.2	<i>User Identification Model</i>	91
8.2.3	<i>Terms-of-Use Model</i>	92
8.2.4	<i>License Model</i>	92
8.3	REJECTION MECHANISM	92
8.4	SESSION ESTABLISHMENT MECHANISM	92
9	CONCLUSIONS	94
9.1	RELATIONSHIP BETWEEN GENERAL BUSINESS PROPERTIES AND BUSINESS FUNCTIONS	95
9.2	PROPOSED GENERAL PHASES AND TRACKS	95
9.3	PROPOSED OPERATIONS	97

9.3.1	Operation: <i>getCapabilities</i> ().....	99
9.3.2	Operation: <i>DescribeProduct</i> (<i>productIDs</i>).....	99
9.3.3	Operation: <i>NegotiatePreConditions</i> (<i>productIDs,conditions, UserID?</i>) .	99
9.3.4	Operation: <i>AgreePreConditions</i> (<i>productIDs,conditions, UserID</i>)	100
9.3.5	Operation: <i>DeliverProduct</i> (<i>token</i>).....	100
10	OUTLOOK.....	102
	BIBLIOGRAPHY	103
	DACS_ACS DTD	104
	DACS_NOTICES DTD	106
	DACS COMMON DTD.....	107
	CUBEWERX NEGOTIATELICENSES XML SCHEMA	108
	TERMS OF USE XML SCHEMA	110

i. Preface

This document is an Open Geospatial Consortium (OGC) IPR for review by OGC members and other interested parties. It is a working draft document and may be updated, replaced by other documents at any time. It is inappropriate to use OGC Draft IPRs (DIPRs) as reference material or to cite them as other than “work in progress.” This is work in progress and does not imply endorsement by the OGC membership.

This document was developed as a deliverable for the OGC Web Services (OWS) 3 Interoperability Initiative as part of the geoDRM Thread Group activity. The authors of this document are also OGC GeoDRM WG members.

ii. Submitting organizations

This Interoperability Program Report is being submitted to the OGC by the following organizations:

- con terra GmbH, Münster, Germany
- CubeWerx, Canada
- DSS, DSS Distributed Systems Software Inc., Richmond, BC, Canada
- Fraunhofer ISST, Dortmund, Germany
- Institut für Geoinformatik (IFGI), Universität Münster, Münster, Germany
- lat lon GmbH, Bonn, Germany
- Metalogic Software Corp., Victoria, BC, Canada
- Universität der Bundeswehr (UniBW), München, Germany
- Traverse Technologies Inc., MA, USA

iii. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

CONTACT	COMPANY
Joshua Lieberman	Traverse Technologies Inc.
Roderick Morrison	Metalogic Software Corp
Cristian Opincaru	University of the German Armed Forces (UniBW)
Ugo Taddei	lat lon GmbH
Roland M. Wagner	con terra GmbH

iv. Revision history

Date	Release	Author	Paragraph modified	Description
2005-11-24	0.2.0	Morrison, Opincaru, Taddei		Implementation Reports
2006-02-06	0.9.1	Wagner		First overall draft
2006-02-06	0.9.2	Wagner		Minor changes
2006-02-09	1.0.0	Wagner		DP proposal
2006-04-19	1.0.0	C. Reed	Various	Fix copyright, header page, edits to preface, forward, and various typos corrected.

v. Future Work

The OWS3 geoDRM activity demonstrated that a number of functional capabilities related to rights management (Terms-of-Use, Authentication, content services) need to be described and chained. The OWS3 geoDRM results are a first step to structure the required processes and to propose solutions.

In a future step, the shown solutions need to be harmonized and the chaining processes need to be described for an engine-to-engine communication processing in detail.

Foreword

After the successful introduction of the Web Map Service Interface Specification (WMS) and other OGC content services, geo-ebusiness related functions have been an increasing focus for OGC standards activities. The OWS3 geoDRM activity is an important step for spatial Rights management by focusing on the (legal) **Terms of Use** for OGC services. The definition is useful, e.g. for disclaimers or to address special user groups with special offers, but using the legal environment to enforce misuse from other user groups.

OWS3 geoDRM was the first step into integrated geospatial rights management into business related functions and into function chaining. It identified many legal and software architectural issues. Because of the complexity of rights management, more initiatives are required to solve interoperability related aspects on different levels.

Although the expression “license” or “click-through license” is often used in discussions, it needs to be considered as a *proposed* license. In the OWS3 discussion, the group showed that the expression “license” may not be correct in this context. The expression “terms of use” is more and more used in a similar context and seems to be more suitable. Many web sites are already using this term. Therefore the expression “Terms of Use” should be preferred. Nevertheless, the usage is not consistent in this report. It is expected that the new draft Abstract Specification topic volume “GeoDRM Reference Model” will define terms more clearly. Therefore the section “Terms and Definitions” is neglected in this document to avoid potential confusion. Only the definition of Terms of Use will be given here as an exception¹:

Terms of Use are rules set up by the owner of an [intellectual property](#) or service to govern how they may be legally used.

In many cases, terms of service are used as a contractual agreement between a company and users of a service they provide. They generally detail restrictions on what each party is and will be responsible for in relation to the service. They may give rules concerning [copyright](#) and other legal details. In a court of law, agreeing to terms of use designates entry into a written [contract](#) in most cases. Where intellectual property is concerned, Terms of Use may be set up in order to let an audience know specifically what can and cannot be done to the work with or without the creator's permission. For written work, terms of use may say that it cannot be distributed by email or other means. Artistic works may stipulate a requirement for compensation in the way of payment, advertising, artist credit and/or other items or services. Musicians may stipulate that they do not allow unauthorized recording of live performances or distribution of such recordings. In [copyright infringement](#) cases, the court may consider the tangible, written terms of use of

¹ http://en.wikipedia.org/wiki/Terms_of_use

the creator in determining whether or not a use of copyrighted materials was legal. It is, therefore, extremely important that terms of use be as specific and accurate as possible.

Introduction

The topic of geospatially enabled Digital Rights Management is very broad. OWS-3 is taking a first step in applying Digital Rights Management to address the unique requirements of geospatial data providers.

The GeoDRM Working Group has identified six classes of GeoDRM use cases (See table 1). OWS-3 addressed two of these six areas, the digital rights models and Authentication & Authorization. The WMS Click-Through Use Case below describes the scope of these initial efforts. It is intended that OWS-3 will provide a foundation upon which more complex GeoDRM capabilities can be built.

Register & Discover - Quickly discover free, rights-restricted, fee-based products (Data and Services)
Describe – Digital Rights & Copyrights, Price Models, Access Rights, Usage Models
Authenticate & Authorize – A registered User, secure transportation with encryption
Price & Order – full-automated procurement of GI products for usage in own applications
Merge – GI sources without losing all geoDRM Data
Use – commercial Products quick & easy by using license categories (a la GI-”GNU”)

Table 1: Identified GeoDRM use case classes

Many of the capabilities needed for this thread have been developed in previous OGC initiatives, other standards bodies and industry consortia.

Previous related OGC efforts include:

- CIPI 1.2 - Critical Infrastructure Collaborative Environment (CICE) – Privilege Management Interoperability Program Report 03-077
- CIPI 1.2 - OGC Distributed Access Control System (DACS) Interoperability Program Report 03-038

The objective of the GeoDRM thread of OWS-3 is to extend the “click-through” licensing concept for web sites to geospatial data services. In particular, click-through licensing techniques were developed for the Web Map Service and Web Feature Service. This activity was coordinated with other OWS threads.

This work developed a first draft of a Geo-Digital Rights information model. The scope of this model for OWS-3 was to describe the components of a license governing the use of a geospatial data set. In this initial phase an unstructured text field was used to define terms of use. Data and service provider are free to define the (legal) content from their point of view and consider their (legal) environment.

The Terms of Use model and the developed service implementation were chained with an authentication mechanism and with a WMS. The policies to be enforced were fairly basic. Data layers are to be made available to users once they have received and accepted the terms of use text governing that data layer (authorization). Specific cases to exercise are:

- User accepts terms of use and receives data
- User rejects terms of use and cannot receive data
- WMS blocks users' access to data under any circumstance

1 Relationship to Other Activities

After the formation of the OGC Geospatial Digital Rights Management (GeoDRM) Working Group (GeoDRM) in June 2004, some general functions have been identified as geospatial business services (authentication, pricing & ordering, license management) and taken into the short-term scope.

Because the OWS3 geoDRM Thread was a sponsored activity, the sponsors could define interoperability tasks that should be solved. The geospatial data providers identified the need to manage Terms of Use. The definition of Terms of Use together with a business and legal environment is a powerful marketing instrument to access user groups in a very distinguish way. Not all rights are reserved or protected. Some may be released to defined user groups. Examples are given for rights related to private and research use, but not for commercial use.

Because currently no other OGC WG is closely working the integration of rights management as part of business functions, the result of the OWS3 geoDRM thread will be carried on by the GeoDRM WG.

2 Use Cases

This section describes six use cases for a better understanding of the context in which Terms of Use are useful. Because the definition and acknowledgement of Terms of Use is a legal act, the business and legal environments are important and need to be considered. Also different business models with different interests will consider some items differently. Nevertheless these use cases are useful to derive requirements for the development process. The complexity is growing from use case 1 to 6.

2.1 Use Case 1: Anonymous User

Terms of Use can also apply to unknown users. In many cases, the data and service provider has little interest to identify users. An example is the usage of a web application. The required special treatments of personal data set are reasons not to know a users identity².

An unknown user is surfing the web in an Internet Café in Münster and interacts with an application (client) accessing spatial data via an OGC WMS services. Prior access the data provider would like the user to read and acknowledge given Terms of Use for the service and its data sets.

The provider informs the user that the data is not accurate enough to use it for navigation purpose. Therefore prior the data access a new WWW browser window pops up with a text field and two buttons “accept” “not accept”. This mandatory interaction should happen only once. After the users’ acknowledgement, he could use the application and services without any re-acknowledgement.

Refinement 1.1: each WMS layer may have own Terms of Use.

Refinement 1.2: A user should need to acknowledge only new (unknown) kinds of Terms of Use.

2.2 Use Case 2: Anonymous User of Remote Service

A business traveller is looking for travel arrangements on various web pages to find suitable hotels. An important criterion is the geospatial distance to his meeting point. The business “Hotelfinder inc.” is offering a mapping service within the hotelfinder portal. Because the hotelfinder business has no interest to handle geospatial basis data, it uses a service of the mapping agency “basisdata4you”. Because of the B2B relationship between the “hotelfinder inc.” and the “basisdata4you”, a contract is signed. No payment is required. A part of the contract declares that the Terms-of-Use may develop, but the hotelfinder application will be informed electronically in advance. Therefore an initial token is created for the hotelfinder inc. If new Terms of Use are in place, the

² Statement: geoconnections

acknowledgement can be done digitally by the legal representative and an updated Terms of Use token will be exchanged. Therefore an evolution process is supported digitally.

2.3 Use Case 3a: Named User

Frequent users of services and applications should be supported by storing their acknowledgements for future use. In this use case, the business model offers frequent users the option to create a log-in with a nick name. The log-in is only required to store this context data. Therefore the frequent user Robert Mayer creates the login “monkey99” to avoid the annoying permanent re-acknowledgement prior each session. He can create his log-in within the applications and can use it instantly.

Refinement 3.1.: Although the service can be used for free, the data provider has an interest to get re-acknowledgements once a week as a kind of reminder.

Refinement 3.2.: Sometimes, the user “monkey99” would like to use different clients, e.g. his GIS for professional use and a simple web client to access the WMS service. Of course, he does not want to maintain two accounts...

2.4 Use Case 3b: Named User with Proof

For some applications and data sets, the business model requires proven identity. Therefore the user can not create a login. He has to express his wish by paper or other classical methods. After the provider grants the log-in account, the user can use it.

Refinement 3.3: see 3.1.

Refinement 3.4: see 3.2.

Refinement 3.5: To reduce maintenance efforts, the user can create an account, but the account is initially disabled. The data provider can activate the account, after suitable proofs of user’s identity were successful.

2.5 Use Case 4: Service Chaining - Out of Band Negotiation

The user Robert requests data sets of a content service. Because he did not agree to the given Terms of Use, the content service refuses the desired delivery. Instead it issues a demand with an appropriated URL, which points to a not standardized web site for humans. Robert acquires a valid token after he interacted and agreed to the terms at the HTML web site. Figure 1 depicts this process.

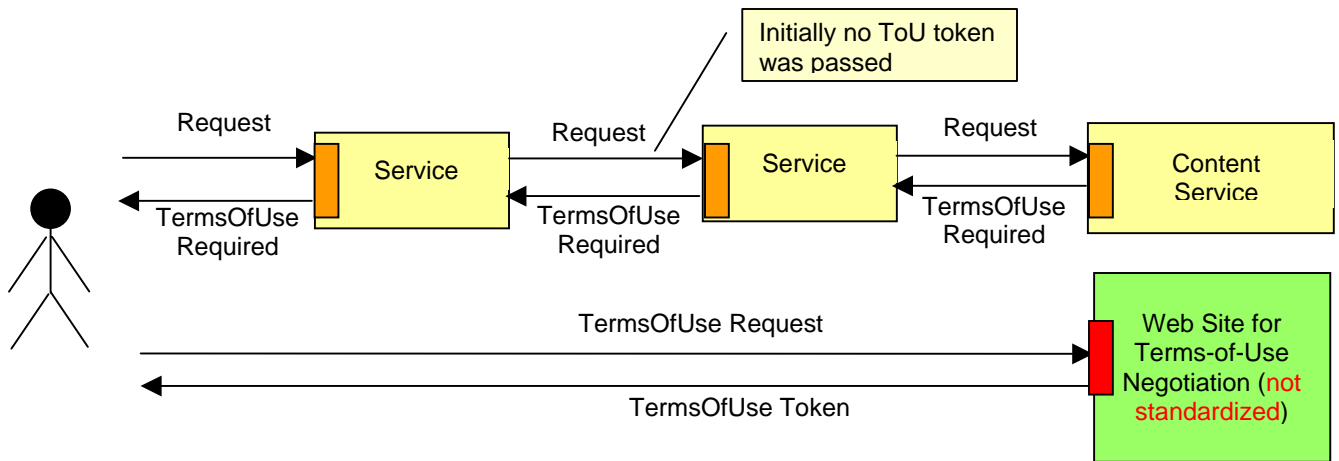


Figure 1: Service Chaining - Out of Band TermsOfUse Negotiation

2.6 Use Case 5: Service Chaining - In-Band TermsOfUse Negotiation

The company “ThePipeLineBuilder” needs many and very different data sets to plan pipelines through Eurasia. Therefore a sophisticated and full automated management of digital rights is required. The price of data sets is less critical than long enduring negotiations and manual processing. The INSPIRE SDI set up harmonized Terms of Use (ToU) categories with standardized names and identifiers. The terms can be obtained in different languages, but reflect the same contracting and legal semantic. The templates are engine-readable. They can be (auto-) filled, e.g. for address information or customer ID. The templates may also contain conditions, which can be (auto-) specified according to the configuration or to user specific context limits. The ToU categories can be ranked. For some categories automated contracting is suitable. Some categories require human interaction. Figure 2 illustrates the value chain.

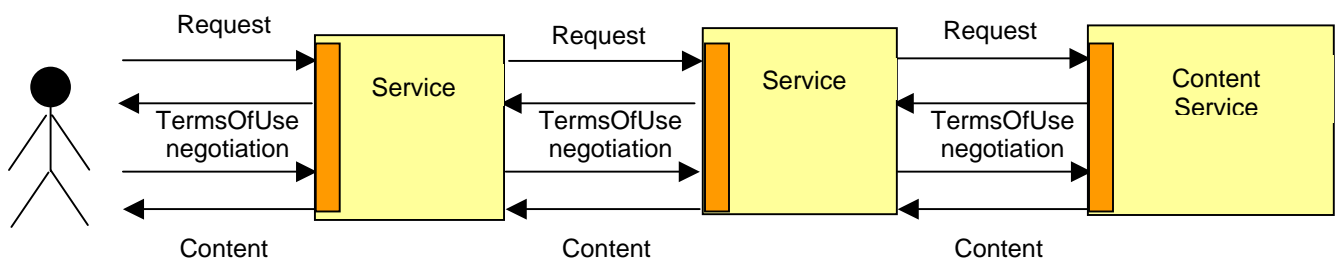


Figure 2: Service Chaining - In Band TermsOfUse Negotiation

Refinements: The negotiate mechanism also contains pricing. The auto-ordering can contain budget limits, e.g. “under 100 EUR”, “order above only with department head agreement” or “budget account x”, which reflects large organizations needs.

2.7 Use Case 6: Service Chaining - In-Band TermsOfUse (multiple cascading)

Use case 5 can also be augmented to tree structures. Multiple providers are offering different datasets in SDIs. Some might offer the same product, e.g. aerial images, but under different conditions (newer, cheaper, better term-of use categories,...). Many business models also prefer redundancy to lower operation and other risks.

If Terms of Use are standardized in categories, the infrastructure can reduce the complexity and amount of contract details. In the case of the “ThePipeLineBuilder” company, geographically different data set might be requested and provided by multiple municipalities, but offered them under the same legal conditions & policies. Therefore a single acknowledgement is sufficient to manage procurement for all data providers.

Figure 3 depicts the use case.

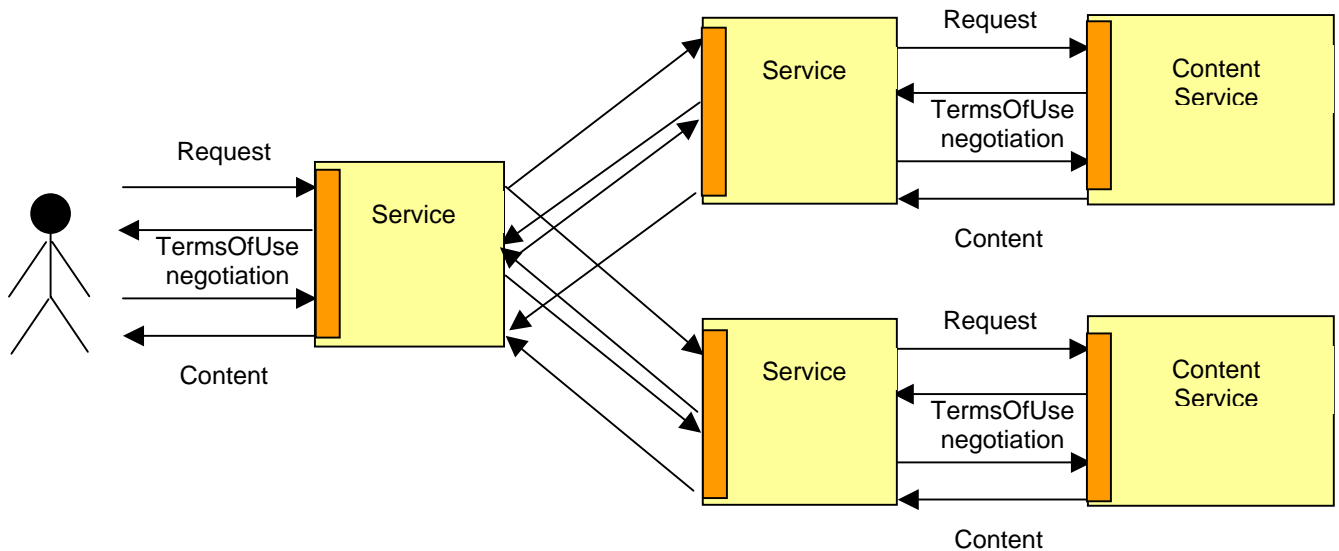


Figure 3: Service Chaining – In Band TermsOfUse Negotiations

The advantages of an interoperable integration of different data sets were shown by WMS services. Analogue to the WMS example, use case 6 assumes also an automated integration of contract elements to reduce complexity for end users.

3 Thread Requirements

This section describes the requirements as defined by the sponsors for the GeoDRM thread. Others are derived from the analysis of described use cases.

3.1 Integration of new functionalities (e.g. terms negotiation) with existing content functions

The use cases show that new functionalities are required to handle the desired business interaction. The new functionalities need specific information models but also new sub processes. On the other hand some components, e.g. web map services (WMS) or web feature services (WFS) are already specified, developed as products and are up & running.

Therefore an integration method is required, which is able to respect existing infrastructures. A general method is the “*Embedding-without-Touching*” approach. A corresponding solution will help to find more provider acceptances for new investments, if not all components need to be upgraded (and paid).

3.2 Session Management

Although the HTTP protocol as a basis of the WMS and WFS specification is state-less, the use cases and the sponsor require a mechanism, which stores the acknowledgement at least for the session between the user and its interface (see use case 1). This requirement may be solved by multiple solutions, which may or may not have an impact for service specifications. Figure 4 shows the interaction in a web client scenario. A user interacts with a regular WWW-browser. The browser interacts with the client. Because of limited browser capabilities, the client may have a browser site part (HTML, JavaScript) and a web server-site part (e.g. Java or C++). This client interacts with the service via a standardized and OGC specified interface.

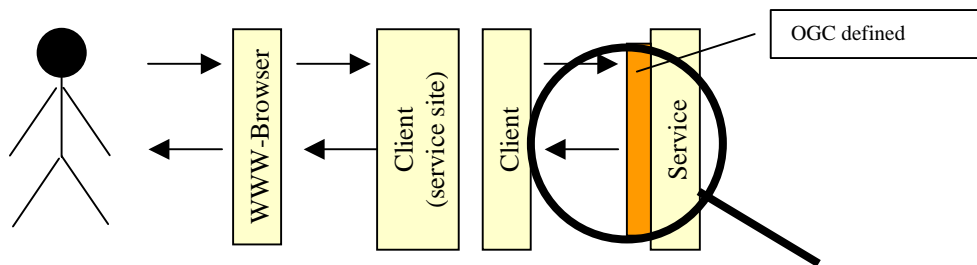


Figure 4: Web Client and Service and interoperability focus

3.3 State Management

The acknowledgement of Terms of Use should only be done once. Therefore states need to be managed. Different solutions with advantages and disadvantages are possible. Figure 5 shows the management of states at the service site.

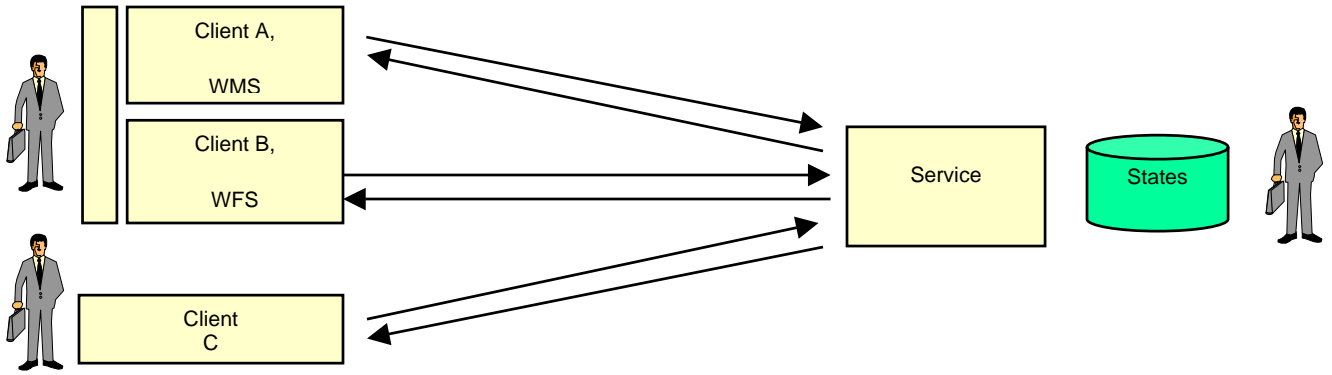


Figure 5: Management of states at service site

A user can use multiple clients with different types and can still access the resources with the same account and conditions. The management of states at the client site is shown in figure 6. The advantage is that a user may define his profile and automate some parts of negotiation. An example is that a user would like to auto-accept specific Terms of Use, e.g. disclaimers. The use of profiles has an advantage in large SDIs with multiple data providers and market conform rules (see use case 4-6).

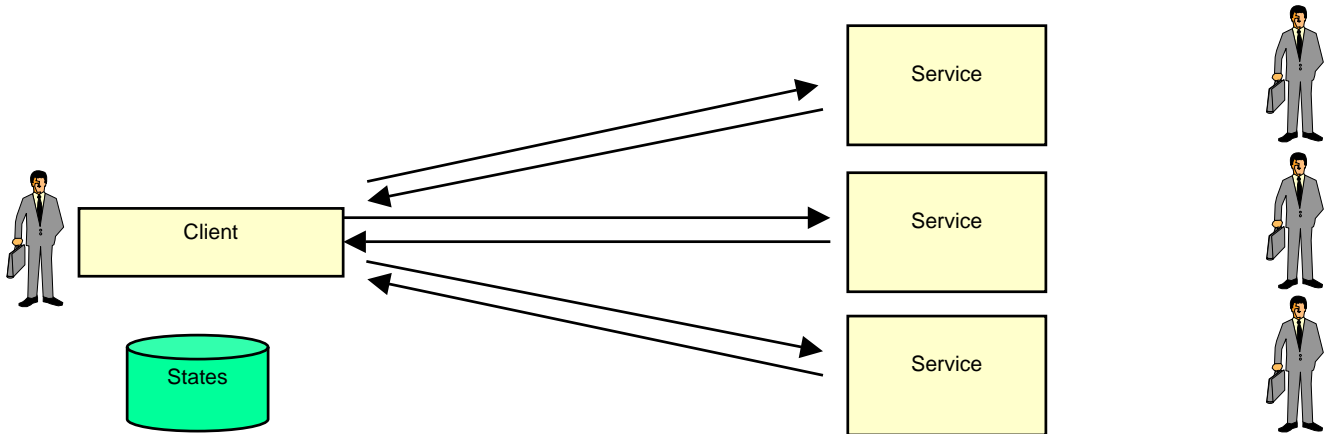


Figure 6: Management of states at client site

It is also possible to manage the states of agreed Terms of Use at both sides.

3.4 Access Control

User-specific requirements are results of the analyses of use cases 3 to 6. To some extent also the anonymous use case 1 can be solved with a “temporal user” authentication management. The solution should respect the described integration requirements (3.1). A security solution should support thick and thin clients, e.g. a GIS and a web client.

3.5 Fully-informed and Trail & Error approach

Terms of Use may be considered as legal transactions. Depending on the content of the text field, there might be a serious impact. Therefore there should be a way to retrieve the conditions in an explicit way prior any (legal) transactions and without any time limits. This process is called “fully informed” process. An approach could be the description in the capabilities document.

The mechanism might be also used to notify a user about new topics in a very general way. In this case a trail & error approach is sufficient. This process design tries to access a service, but is prepared to be thrown back with an error, if the credentials are not sufficient.

Both approaches are valid. The specification should support both approaches to support a wide range of business cases.

3.6 Different HTTP Technologies: Get, Post and SOAP

Because of the different underlying transport technologies, there might be different suitable implementation solutions. Although it would be desirable to have the same abstract information models and process models, which could be derived and encoded in different ways for the different HTTP technology platforms.

3.7 Explicit and implicit description and processes

The degree of explicit description is a relevant factor for interoperability, because it is reducing potential misunderstandings. A minimal implicit approach is just an unstructured ID. The advantage is that it could be used in various forms within a known community. A more structured approach is the definition of operations and parameters. This reduces misunderstandings in not known communities. The WMS specification is an example for an explicit specification.

3.8 Backwards compatibility

The Geo-eBusiness Terms-of-Use functionality can be considered as orthogonal to any content and processing services. Many business models may still use older versions of interoperable OGC products and its specifications. The installed software products are up and running. An upgrade just because of the Terms-of-Use function itself may not be considered by many operators.

3.9 Different Packaging of new business functionality

The new business functionality may be packaged differently by software producers depending of their product range. The following variants (stand-alone and fully integrated) were already identified and have some characteristics. The specification must support at least the both identified variants.

3.9.1 Stand-alone Variant

The new business functionalities are packaged as a stand alone component and integrated within the protocol stream like proxies. This variant has the following advantages:

- existing web services and established user relationships could be upgraded without changing content services
- Easier network administration in professional environments with multiple networks and firewalls
- Support of multiple content services with common business functionalities and therefore less management efforts, e.g. 1 ToU service for 10 WMS, because the ToU apply to all 10 WMS
- No or little dependency between software products from different vendors and therefore more flexibility for business developments

In this case the component needs to be considered as a self describing service.

3.9.2 Fully integrated Variant

Another possible package in the integrated variant, which integrates business functions with content functions natively. This packaging approach has the following advantage:

- Higher performance due to native APIs
- No integration nor configuration efforts

4 Implementation: Cubewerx/DSS/Metalogic

The CubeWerx/DSS/Metalogic team proposed an implementation based on DACS – the Distributed Access Control System (<http://dacs.dss.ca>). Under the OWS3 GeoDRM initiative DACS access control has been extended to enforce notice acknowledgement constraints. The anonymous click-through license identified by the project sponsor as the key deliverable has been implemented as a special case of this work. CubeWerx (<http://cubewerx.com>) has extended their OGC WMS products CubeXPLOR and CubeSERV to interoperate with DACS authentication and notice acknowledgement and to support more “fine-grained” geospatially-informed license acknowledgement (*e.g.*, a GetMap request within a specified geographic region, or for a particular WMS layer or layers may require that a license must be acknowledged).

4.1 DACS Overview

DACS is a general purpose framework for control of access to web resources implemented in an Apache 2.0.x module and a suite of CGI programs and web services. It can be used as a universal authentication mechanism for a single Apache server or as a full-fledged, single sign-on multi-server identity management and access control system. DACS is available on SourceForge (<http://sourceforge.net/projects/dacs>) under a dual open source/commercial license similar to that of Berkeley DB.

ACS has been described by some as a “look-aside” architecture because, for each HTTP request received by the Apache server for a resource under a “DACS-wrapped” location, a database of access control rules is consulted to determine if access should be granted. If the result of this evaluation is to *allow*, normal Apache request processing is executed. If the result is to *deny*, one of several configurable processes is followed:

1. an HTTP 403 forbidden status is returned in the response (customizable using Apache’s native ErrorDocument directives)
2. a browser redirect to a DACS event handler associated with the reason the request is denied; one of:

Code: 900 Name: NO_RULE Synopsis: Access denied, no applicable rule Description: All rules were examined but no rule applies to the service request
Code: 901 Name: BY_RULE Synopsis: Access denied, forbidden by rule Description: The closest matching rule does not grant the service request
Code: 902 Name: NO_AUTH Synopsis: Access denied, user not authenticated Description: No valid credentials were provided and either a) no rule applies, or b) the rule does not grant the service request

Code: 903 Name: REVOKED Synopsis: Access denied, user access revoked Description: Credentials were explicitly revoked
Code: 904 Name: BY_REDIRECT Synopsis: Access denied, redirect Description: A rule has explicitly redirected the user
Code: 905 Name: ACK_NEEDED Synopsis: Access denied, acknowledgement needed Description: One or more notices associated with the request must be acknowledged
Code: 998 Name: UNKNOWN Synopsis: Access denied, reason unknown Description: An error occurred during processing

4.2 DACS Access Control Service – the ACS Module

The **DACS** access control service (or simply **ACS**) is the component of **DACS** responsible for making access control decisions. It is implemented by the `dacs_acs` program. **ACS** provides role-based access control using access control lists, also called access control rules and ACLs. **ACS** controls access to arbitrary services, which may be resources, such as data or files, or programs.

At present, services are typically web-based and service requests are expressed as URLs. In this configuration, a web server runs **ACS** to determine whether a particular service request is authorized. For the Apache web server, a **DACS**-aware module called [mod_auth_dacs](#) interacts with **ACS**. A web server having [mod_auth_dacs](#) functionality is said to be *DACS-enhanced* and web services that are under the control of [mod_auth_dacs](#) are said to be *DACS-wrapped*.

When a web server receives a **DACS**-wrapped service request, it consults **ACS** to determine whether the request should be granted. The web server provides **ACS** with the name of the requested service ("What is being accessed?"), parameters that were passed in the request ("How is it being accessed?"), the identity of the client ("Who is making the request?"), and other context associated with the request. With this information at hand, **ACS** consults a set of access control rules (the ruleset) (see [dacs.acls\(5\)](#)). Additional contextual information, such as **DACS** configuration directives and build-time options, and the run-time environment, is also available. **ACS**'s response to the web server either grants permission, possibly with a constraint that specifies additional, service-specific information, or denies permission, possibly with a reason for denial. **ACS** may also instruct the web server to redirect the client.

All **DACS** services must be under the control of **ACS**, even those that do not require the client to be authenticated. Also, a web server must be configured such that only **DACS**-controlled services and no other services can be invoked through URLs associated with its **DACS** jurisdiction.

Please refer to the documentation for [mod_auth_dacs](#) for information on configuring the **DACS** Apache module.

4.2.1 Module-to-ACS Protocol

The Apache [mod_auth_dacs](#) module invokes **dacs_acs** to do the hard part of deciding whether a request should be granted or denied. The module is responsible for configuring itself using new Apache directives, gathering information required to make the access control decision, passing that information to **ACS**, and receiving the access control decision from **ACS**, together with either environment information (if access is granted to an executable request) or error handling directives (if access is denied).

To prevent potentially sensitive information from becoming visible, [mod_auth_dacs](#) passes information to **ACS** over a pipe. **ACS** reads its standard input, makes the access control decision, and writes either environment information or an optional error handling directive to its standard output. The exit status of **ACS** communicates its decision: zero means the request should be granted, anything else means the request should be denied.

The information passed *to ACS* is in the format:

```
variable-name="variable-value"
```

each of which is terminated by a newline character.

The current set of variables is listed here:

`SERVICE_ARGS`

The query arguments (if any and whether `GET` or `POST` method is being used) followed by `POST` arguments (if any and to the maximum length configured), base64 encoded.

`SERVICE_ARGS_TRUNCATED`

For `POST` method requests, if the `POST` data stream was not completely captured because the maximum length was reached, this variable will be present and assigned the value 1.

`SERVICE_AUTHORIZATION`

The value of the `Authorization` HTTP header field, if present.

`SERVICE_COOKIE`

The value of the `Cookie` HTTP header field, if present.

SERVICE_CONTENT_ENCODING

The value of the `Content-Encoding` HTTP header field, if present.

SERVICE_CONTENT_LENGTH

The value of the `Content-Length` HTTP header field, if present.

SERVICE_CONTENT_TYPE

The value of the `Content-Type` HTTP header field, if present.

SERVICE_FILENAME

The name of the file, as determined by Apache, corresponding to this response.

SERVICE_POSTDATA

When available, the `multipart/form-data` stream (or part of it), base64 encoded.

SERVICE_HOSTNAME

The name of the host as set by the full URI or `Host` HTTP header field, as determined by Apache.

SERVICE_HTTPS

If the request came over SSL (HTTPS), this variable will be present and set to "on".

SERVICE_METHOD

The request method, as set by Apache (e.g., "GET").

SERVICE_PATH_INFO

The `PATH_INFO` part of the URI, as set by Apache.

SERVICE_SERVER_PORT

The TCP/IP port on which the request was received by Apache.

SERVICE_AUTHORIZATION

The Authorization header, if received by Apache.

SERVICE_PROXY_AUTH

The value of the `DACS-Proxy-Authorization` HTTP header field, if present.

SERVICE_QUERY

The value of the query string component of the URI.

`SERVICE_REMOTE_ADDR`

The client's IP address.

`SERVICE_REMOTE_HOST`

The client's DNS name, if known by Apache.

`SERVICE_URI`

The path portion of the URI, as determined by Apache.

`SERVICE_USER_AGENT`

The value of the `User-Agent` HTTP header field, if present.

If access is granted, **ACS** may provide a set of control directives for [mod_auth_dacs](#) to interpret, followed by a set of environment variables for [mod_auth_dacs](#) to introduce into the environment of an executable request. Each control directive starts with a "=" character and is terminated by a newline. Environment variables are specified in the format:

variable-name=variable-value

each of which is terminated by a newline character.

If access is denied, **ACS** may instead provide an error handling directive, newline terminated, in the form expected as the third argument to Apache's `ap_custom_response()` function.

4.2.2 Credentials

DACS credentials can be passed to **ACS** in several ways, but they have the following representation:

DACS:federation-name:jurisdiction-name:username=value[; ...]

The string is URL encoded. If there are multiple credentials, they are separated by any combination of spaces and ";" characters.

Credentials are passed from [mod_auth_dacs](#) to **dacs_acs** using the `SERVICE_COOKIE` variable (transmitted over a `pipe(2)`)

Because a process's environment is public on some systems, **DACS** takes care not to pass credentials using environment variables. Passing credentials through the `HTTP_COOKIE` environment variable is forbidden unless the `ALLOW_HTTP_COOKIE` directive has the value "yes". This behavior can be overridden if necessary, however. When specifically enabled, they can be passed using the `DACS_COOKIE` environment variable.

4.3 DACS Notice Acknowledgement

The event code 905, ACK_NEEDED, appearing in the list of possible DACS access denied events is a new code introduced under the OWS3 GeoDRM initiative. A 905 event is triggered when access to a Web resource is denied because a required notice acknowledgement token does not accompany the request. A specification for the syntax and semantics of notice acknowledgement tokens was written under this initiative and is included in the appendixes to this document.

The requirement for notice acknowledgement is specified in a DACS access control rule using a new ack() predicate. To illustrate its use, the following example ACL rule expresses that the text in two disclaimer documents must be acknowledged before access to the resource (anything under location /notices-must-be-acknowledges/) will be allowed:

```
<acl rule>
  <services>
    <service url_pattern="/notices-must-be-acknowledged/*"/>
  </services>
  <rule order="allow,deny">
    <allow>
      ack("http://demo.fedroot.com/notices/arjis-disclaimer.html",
          "http://demo.fedroot.com/notices/usgs-disclaimer.html")
    </allow>
  </rule>
</acl_rule>
```

When a request is received -- say for `https://demo.fedroot.com/notices-must-be-acknowledged/index.html` -- DACS examines the request for accompanying NATs. If NATs are found they are decrypted and the NOTICE_URIS component is matched against the notices named in the ack() predicate. If a match is found, access is allowed, else denied.

In the case that access is denied, DACS processes a 905 ACK_NEEDED event. A typical configuration will associate a DACS notice presentation service with the ACK_NEEDED event. This is accomplished in the DACS configuration file using an ACS_ERROR_HANDLER directive:

```
ACS ERROR HANDLER "ACK NEEDED"
https://demo.fedroot.com/fedadmin/dacs/dacs_notices"
```

As a result of 905 event handling the user's browser is redirected to the dacs_notices URL that has been associated with the event. dacs_notices dereferences the contents of the two disclaimer documents and presents them to the user in an HTML form. The user

must indicate acceptance of the documents by selecting the “Accept” radio button and clicking SUBMIT. dacs_notices is invoked again as follows (this time acting as a notice acceptance service):

```
https://demo.fedroot.com/fedadmin/dacs/dacs_notices?  
NOTICE_URI=http://demo.fedroot.com/notices/geobase-license-  
agreement.html+http://demo.fedroot.com/notices/usgs-  
disclaimer.html&  
RESOURCE_URI=https://demo.fedroot.com/notices-must-be-  
acknowledged/index.html&  
TIME=1131988373&  
HMAC=XrC4HDzdGjV44N9Gh.GDnrngfuA&  
RESPONSE=accepted
```

In the request to dacs_notices to accept, TIME and HMAC attributes are sent with the request. These attributes connect the acceptance event with the original request for notice acknowledgement. Acceptance must occur within a configurable time from the original request and the HMAC that is sent must agree (cryptographically) with the function used originally by DACS to generate it.

In addition to the 902 ACK_REQUIRED handler there are two other handlers that may be configured in DACS:

- **NOTICES_ACCEPT_HANDLER:** the URL (absolute or relative) to which a user agent will be redirected after a positive acknowledgement to a notice has been received (i.e., the notice or notices were "accepted"), if it is not possible to redirect the user agent to the request that initiated notice acknowledgement processing (e.g., if that request used the POST method).
- **NOTICES_DECLINE_HANDLER:** the URL (absolute or relative) to which a user agent will be redirected after a negative acknowledgement to a notice has been received (i.e., the notice or notices were "declined").

Note: Of course, apart from answering a skill-testing question or the like, there's no way of knowing that a user has actually read and understood the notices. It is unclear to what extent it is necessary to go in this regard with respect to providing support. **DACS** cannot guarantee that a human user has actually read these notices and indicated acceptance of them, but it can guarantee (optionally) that a NAT cannot be obtained by a client without the client having received a copy of the notices. If the client wants to "trick the system" by not actually presenting the notices to the user or soliciting a response it can, and in this event the service provider might consider legal recourse.

4.4 Middleware Support

dacs_notices can be asked to emit various flavours of XML in support of middleware or thick clients. This is useful when middleware would prefer to prompt the user itself (acting as a notice presentation handler) and then invoke a acknowledgement handler (such as **dacs_notices**) to obtain a NAT. Any customizations specified for HTML output are ignored when XML is being produced and are not passed to middleware. In the OWS3 GeoDRM thread this support was used by CubeWerx to integrate DACS notice acknowledgement and authentication requirements with their CubeXPLOR WMS middleware and by Refractions to integrate these same DACS functions with the GeoDSS client. Details of the CubeWerx development are provided below.

The XML emitted by **dacs_notices** conforms to the DTD [dacs_notices.dtd](#) included in the appendix to this document. When acting as a notice presentation handler, it returns a `presentation_reply` element and when acting as a notice acknowledgement handler, it returns a `ack_reply` element; in either mode of operation an error reply is possible (via the `common_error` element).

In conjunction with [dacs_acs\(1\)](#), **dacs_notices** can optionally operate in a "secure" mode, where a particular control flow is enforced; refer to the `NOTICES_SECURE_HANDLER` directive in [dacs.conf\(5\)](#). The non-secure mode will be described first.

4.4.1 Simple Mode

The `presentation_reply` element lists one or more notices that must be acknowledged by the user. It includes a space-separated list of the URIs of the notices and a space-separated list of the URIs of resources that require these notices to be acknowledged. The text of the notices are base64 encoded within the `notice` element, as specified by [RFC 2045](#) (Section 6.8). The notice's URI is included as an attribute.

The `ack_reply` element returns the user's response and, optionally, a URI to which the user can be redirected (depending on the context, this may be the URI of the request that required notices to be acknowledged, the value of the `NOTICES_ACCEPT_HANDLER` directive, or the value of the `NOTICES_DECLINE_HANDLER` directive). If a NAT is issued, it is returned (as an HTTP cookie) by the notice presentation handler.

4.4.2 Secure Mode

The secure mode of operation, which may not be necessary in some environments, serves two main purposes:

1. a NAT can be cryptographically protected against forgery and alteration. Refer to [dacs.nat\(5\)](#).
2. **DACS** can enforce a flow of control so that a client cannot obtain a NAT without having received a copy of the notice(s); this is the purpose of the `hmac` and `time`

attributes. That is, **DACS** can make it necessary for the client to first call **ACS** with *-check_only* or *-check_fail*, then call the notice presentation handler to get the text of the notices, and then call the notice acknowledgement handler to request the NAT. No other control flow is possible (ignoring errors).

When combined, these protections make it difficult for an attacker or unfriendly user to bypass having to acknowledge notices by manufacturing NATs or having **DACS** simply issue arbitrary NATs.

Regardless of the selected output format, the required flow of control is:

1. **DACS** ACS receives a service request
Access to the requested resource will not be granted by [dacs_acs\(1\)](#) until one or more notices have been acknowledged by the user. **ACS** either redirects the client to the notice presentation handler or returns an XML document ([dacs_acs.dtd](#)) that describes which notices must be displayed and acknowledged; the behaviour depends on the service request. The notice presentation handler must be invoked and will expect to be passed the *hmac* and *time* arguments.
2. Notice presentation handler is invoked
The user is expected to be presented with the notices and asked to accept or decline them. The handler either returns a web page containing an HTML `form` or an XML document ([dacs_notices.dtd](#)). In either case, the handler will verify that **ACS** had been called "recently" (the security-related arguments expire after a set amount of time and cannot be reused). Its output will include *hmac* and *time* arguments, either of which may differ from the values passed into the program; the notice acknowledgement handler expects to be passed these arguments.
3. Notice acknowledgement handler is invoked
The user's response is directed to the notice acknowledgement handler, which verifies that the notice presentation handler has been called. The handler either redirects the user appropriately or returns an XML document ([dacs_notices.dtd](#)). If no error has occurred and the user has accepted the notices, a NAT will also be returned.

4.5 Implementation of a GetUnsatisfiedPreconditions Service in DACS

The OGC specification for WMS contemplates two possible mechanisms for signalling that an exception has occurred: `EXCEPTION=INIMAGE` which embeds an image rendering of a text message and the default behaviour which returns an exception document satisfying an XML DTD that is defined as part of the OWS Common Architecture initiative (RNM: is this correct?). This exception mechanism allows a WMS client to react reasonably to exceptions that are returned from a WMS server.

But what happens when a generic access control mechanism is inserted between a WMS client and server? The response that a WMS client may receive as the result of an access control exception is not part of any OGC specification.

In situations like this client applications and middleware need a means to determine if unsatisfied preconditions (*e.g.*, user must be authenticated, notices must be acknowledged) exist that would prevent a service request from being fulfilled, so that these conditions may be negotiated by the client prior to issuing their own native requests to the server.

To support this requirement DACS was extended under the OWS3 GeoDRM initiative to implement a GetUnsatisfiedPreconditions service, which formed the basis for work by CubeWerx and Refractions on the integration of DACS with their WMS client applications.

4.5.1 Testing Access

There many situations in which it is sometimes important for an application or middleware to know whether **DACS** will grant or deny a service request without having to actually execute the service request. For example, when building a menu, an application might want to exclude items involving service requests that would be denied to the user. The DACS **ACS** module provides this functionality. In some situations, if access would be denied **ACS** will return an indication of what must be done; *e.g.*, the user must authenticate or a notice must be acknowledged. Note that there can be multiple reasons for denying access, in which case an application may have to repeatedly request a check and address the reason for denial before access may be granted.

To check whether access would be granted or denied, the application invokes the **DACS**-wrapped service or resource *exactly as it would normally except that it must provide an additional argument named `DACS_ACS`*. The value of this argument is parsed like a set of space-separated command line flags. Note that the space character(s) must be properly escaped; *e.g.*, as `%20`. The following flags are recognized:

`-check_only`

The presence of this argument tells **ACS** not to actually execute the web service or return the resource, but to merely return the access control decision. This flag and the `-check_fail` flag are mutually exclusive.

If the access check was performed, HTTP status code 200 (OK) will be returned; any other result indicates that the check could not be executed (*e.g.*, due to an Apache configuration problem or a **DACS** error). If the check is performed, the default response consists of a single line of text that gives the result, The line consists of a three digit result code, followed by a space, an explanatory message, and a newline character:

```
797 Access denied
798 Access granted
799 Access error
```

Inspecting the result code is sufficient to obtain the outcome of the check. Any Apache ErrorDocument directive for "error-code" 200 is overridden. The `-format` (see below) can be used to select a different output format.

Note that the service or resource in question does not have to exist for **ACS** to grant access; this can happen if a wildcard rule pattern is used. Also, keep in mind that access control rules can be written to be highly context specific; there is no guarantee that the same decision made at one point in time will also be made an instant later (access control rules can depend on the current date or time, for instance).

Rules can be written such that their evaluation results in persistent changes; for example, a database might be updated. These kinds of changes will occur both in normal operation and when only checking access. **ACS** defines the variable `${DACS: :ACS}` only during the testing mode of operation so that, if necessary, rules can be written to differentiate between testing mode and normal operation.

`-check_fail`

Like the `-check_only` flag,

except if access is granted the request is allowed to proceed. The current implementation is arguably buggy in that the `DACS_ACS` argument is visible to any program invoked in this way; flags should be added to specify whether it should be erased or retained. This flag and the `-check_only` flag are mutually exclusive.

`-format fmt`

By default, the `-check_only` flag (and in the case where access is denied, the `-check_fail` flag also) results in a single line of text being output (equivalent to "`-format text`"). If more detail is required, an XML description can be produced by specifying any of the XML output formats (refer to [XML Output](#), the [FORMAT CGI argument](#), and the `-format` command line argument).

If some part of the `DACS_ACS` argument is invalid, the initial valid part will still be effective; e.g., if the initial part is `-format XML`, the output format will always be XML.

Assuming the target resource is **DACS**-wrapped, instead of returning the resource, accessing the following URL would return an indication of whether an actual request to access the resource would be granted or denied:

https://fedroot.com/foo.html?DACS_ACS=-check_only%20-format+xml

ACS removes the `DACS_ACS` argument from its environment so as not to disturb access control processing. It is not possible to escape this argument. **DACS** credentials may accompany the service request just as they would a real request and are incorporated into the check.

4.5.2 XML Output

When XML output has been enabled, **ACS** will emit a document (conforming to [dacs_acs.dtd](#) included in the appendix) when access is denied, a processing error occurs, or when an access testing mode has been requested using the `DACS_ACS` argument.

ACS associates an error code with each event or reason for which access might be denied (see the description of the `ACS_ERROR_HANDLER` directive in [dacs.conf\(5\)](#)). The error code is itself sufficient for a client to know why access was denied. When access is denied, an appropriately named XML element is emitted. The element will include an explanatory text message, and optionally, the URI of a handler that the client might call to continue the workflow. This URI is obtained from the applicable configured `ACS_ERROR_HANDLER` directive, if any.

The `event905` element corresponds to the `ACK_NEEDED` (equivalent to error code 905) **DACS** error event. It is emitted if the client must acknowledge one or more notices before the request will be granted. Its handler attributes, which are optional, are obtained from the `ACS_ERROR_HANDLER` directive that applies to this error and the `NOTICES_ACK_HANDLER` directive. If the `ack_handler` attribute is absent, then the `presentation_handler` is expected to perform both presentation and acknowledgement handling functions. The `notice_uris` attribute is a comma-separated list of URIs of notices that must be acknowledged by the user. The `resource_uris` attribute is a comma-separated list of URIs of resources associated with this request; this will usually be only a single URI. The `time` and `hmac` attributes are used to enforce a secure workflow mode. Please refer to [dacs_notices\(1\)](#) and [dacs.conf\(5\)](#) for additional detail.

A `common_status` element indicates that **ACS** could not process the request. This might happen, for example, if `dacs_acs` were not properly configured.

4.5.3 Identity interoperability

One of the key challenges facing the OWS3 GeoDRM Thread are the significant architectural differences between the various vendor solutions.

DACS currently supports a wide variety of authentication methods that are oriented towards establishing a DACS identity by interacting with a human user (*e.g.*, AD/LDAP, /etc/passwd, Apache native authentication, PKI certificates, *etc.*). It is sometimes useful for an external identity management system to request DACS credentials. Whether it is a "foreign" (non-DACS) system or simply a different DACS federation, such a feature would allow any trusted entity to obtain DACS credentials. This could be used, for example, to convert a foreign system's native identity to a DACS identity. Of course these requests would be subject to DACS access control.

DACS already has a web service that will "decode" DACS credentials, allowing foreign systems to work with DACS identities. As part of the OWS3 initiative a new `dacs_auth_agent` service (see [dacs_auth_agent\(1\)](#)) was implemented to provide complementary functionality, providing a way for foreign systems to obtain DACS

credentials. Just like DACS credentials obtained by users through a login procedure, these DACS credentials could subsequently be used to access DACS-wrapped resources.

4.5.4 `dacs_auth_agent`

The `dacs_auth_agent` web service is used to request **DACS** credentials outside of the usual **DACS** authentication procedure (see [dacs_authenticate\(1\)](#)). The client making the service request, whether a user agent or middleware, is considered to be an "agent" trusted by the jurisdiction that receives the request by virtue of having obtained **DACS** credentials and satisfying **DACS** access control rules that grant it access to this service. Access control rules are responsible for expressing restrictions on the types of operations to be granted to various trusted agents.

The client's **DACS** credentials can be obtained through [dacs_authenticate\(1\)](#), [dacscred\(1\)](#), [cookie\(1\)](#), or even `dacs_auth_agent`.

If the request is successful, credentials are returned to the client within an HTTP cookie. Credentials generated by this service can be distinguished from those created by one of the other methods.

4.5.5 Warning

Access to this web service must not be granted without establishing and testing carefully crafted access control rules and appropriate configuration. By default, access to this service is always denied.

Much like Unix's `su(1)` command lets the `superuser` assume the Unix identity of any other user, this service provides a way for a privileged client to request credentials for a user known to the receiving jurisdiction. Any other credentials in the possession of the client remain in effect.

The service can also be invoked to effectively import an identity that is recognized by the agent but possibly not known to the receiving jurisdiction. This provides a way to convert foreign credentials, whether from a non-**DACS** based system or a different **DACS** federation, into credentials understood by the federation of the receiving jurisdiction. It is only necessary for the agent to understand the foreign credentials; **DACS** never sees them.

Another use of this service is in conjunction with middleware that does its own authentication. Having authenticated a user, an application can ask `dacs_auth_agent` for **DACS** credentials for the user.

`dacs_auth_agent`, combined with the existing `dacs_current_credentials` service and the **DACS** `cookie` command provide numerous options for foreign systems to interoperate with **DACS** identities.

4.6 CubeXPLOR-DACS-CubeSERV Work Flow

4.6.1 Introduction

This section describes work carried out by CubeWerx under the OWS3 GeoDRM thread to integrate CubeXPLOR and CubeSERV with coarse-grained (DACS-level) access control and a fine-grained (CubeSERV-level) license mechanism. The description is presented from the perspective of server-side middleware that is acting as a mediator for WMS services (e.g., CubeXPLOR). Thicker clients (that make GetMap requests directly rather than returning an image reference to a browser) can avoid the CubeSERV fine-grained NegotiateLicenses step by making a GetMap request with EXCEPTIONS=XML and only performing license negotiation if the response is an UnsatisfiedLicensesExist exception.

4.6.2 The start of the Workflow

The first thing CubeXPLOR does, as usual, is perform a GetCapabilities request to CubeSERV. It is assumed that DACS has been configured to allow this request without a license requirement. The capabilities document returned will indicate which, if any, license mechanisms are employed by the server. They are indicated in the `<AccessConstraints>` element as a comma-separated list of options.

The options (which may co-exist) are:

- DacsGeoDRM - coarse-grained (DACS-level) access control mechanism (including notice acknowledgment and authentication)
- CwGeoDRM - fine-grained (CubeSERV-level) license mechanism

CubeSERV determines whether or not it is under DacsGeoDRM control by examining the `DACS_CONSTRAINT` and `DACS_DEFAULT_CONSTRAINT` environment variables for the presence of the "do-license-management" constraint. These variables are set in DACS ACL rules configured to control access to CubeSERV.

4.6.3 Coarse-grained License Management

If the CubeSERV capabilities document indicates that a coarse-grained (DACS-level) license mechanism is in effect, CubeXPLOR must then ask DACS to perform a license check for each (or, at minimum, the first) of the required GetMap requests. If not, then skip to the section entitled "FINE-GRAINED LICENSE MANAGEMENT".

A GetMap request is checked by sending the request with an extra "DACS_ACS=-check_only+-format+XML" parameter to CubeSERV as described in the section above describing the DACS implementation of the GetUnsatisfiedPreconditions service. DACS will intercept this request because of the `DACS_ACS` parameter; thus, CubeSERV will never receive it. The current DACS NAT cookie (which has encoded within it the list of already-accepted licenses) is sent along with the request. CubeXPLOR expects a `dacs_acs` XML document as a response. The DTD for `DACS_ACS` is included in an

appendix to this document. If there are no unsatisfied licenses for that GetMap request, the document returned is simply:

```
<dacs acs>
  <access_granted/>
</dacs_acs>
```

However, if there are unsatisfied licenses for that GetMap request, a dacs_acs response document like the following is returned:

```
<dacs acs>
  <access_denied>
    <event905
      presentation_handler="http://...presentationHandler..."
      ack_handler="http://...ackHandler..."
      notice_uris="http://...+http://..."
      resource_uris="http://...+http://...">
      <notices>
        <notice_uri uri="http://..."/>
        <notice_uri uri="http://..."/>
      </notices>
    </event905>
  </access_denied>
</dacs_acs>
```

Upon receipt of such a response, CubeXPLOR sends an HTTP redirection back to the browser, redirecting the browser to the stated presentation handler, with the following parameters:

```
http://...presentationHandler...?
RESOURCE_URIS=http://...+http://...
&NOTICE_URIS=http://...+http://...
&DACS_ERROR_URL=http://...cubeexplorCallbackUrl...
```

where the value of DACS_ERROR_URL is the CubeXPLOR URL that the presentation handler should return the browser to once this stage of license management is completed. It is assumed that this presentation handler will prompt the user to accept the required licenses, that it will automatically call the ack handler, and that a new DACS NAT cookie (containing an updated list of already-accepted licenses) will be returned to the client before the client is redirected back to CubeXPLOR.

CubeXPLOR should perform this for each of the required GetMap requests. Note that the use of cookies to implement client-side state means that the coarse-grained access control mechanism requires both CubeXPLOR and CubeSERV to be in the same DACS jurisdiction, i.e., to have the same domain name. Other options such as custom headers and URL-rewriting are possible with DACS but these were not pursued in this initiative.

4.6.4 Fine-grained License Management

Once the coarse-grained license management is completed (or if no coarse-grained license management was required), and if the CubeSERV capabilities document indicates that a fine-grained (CubeSERV-level) license mechanism is in effect, CubeXPLOR must then ask CubeSERV to perform license negotiation on the required GetMap requests. (If no fine-grained license mechanism is active, then skip to the section entitled "THE ACTUAL GETMAP REQUEST".) Unlike with the coarse-grained license mechanism, this can be done with a single request that tests all of the required GetMap requests at once.

This is done with an HTTP POST NegotiateLicenses request. (The NegotiateLicenses schema is included as an appendix to this document.) A key-value-pair HTTP GET version of this request is defined as well. However, in practice it is better to use the HTTP POST version where possible because the potential size of the HTTP GET URL might be problematic.

CubeXPLOR maintains a table that specifies a 1:1 mapping from server URLs to accepted-licenses tokens. This table is cached on the browser by means of a vendor-specific cookie called CWLICENSES. This allows any user of that CubeXPLOR to connect to different CubeSERVs without requiring the CubeSERVs to be in the same domain name as the CubeXPLOR. When CubeXPLOR makes a NegotiateLicenses request to a particular CubeSERV, it includes that server's accepted-licenses token as the value of the <Accepted> element.

Here is a sample response:

```
<NegotiateLicensesResponse>
  <Accepted>bmV3Zm91bmRsYW5kV2FybmluZ6dnb29nbGVE</Accepted>
  <UnsatisfiedLicenses>
    <License id="uniqueId42">
      This is the text of license "uniqueId42".
    </License>
    <License id="uniqueId99">
      This is the text of license "uniqueId99".
    </License>
  </UnsatisfiedLicenses>
</NegotiateLicensesResponse>
```

When CubeXPLOR receives a NegotiateLicensesResponse, it first takes the value of the <Accepted> element, updates its table accordingly, and makes sure that the new value of the table makes it to the browser's cache (by eventually sending an HTTP Set-Cookie directive to the browser).

If there are no UnsatisfiedLicenses listed, then license negotiation is complete. Otherwise, CubeXPLOR generates a web page containing a license-acceptance form which displays the provided license texts (and internally remembers the corresponding license IDs). When this form is submitted (with everything ticked as being accepted, of course), CubeXPLOR then sends another NegotiateLicenses request, this time specifying (by means of the <JustAccepted> element) the list of license IDs that were just accepted

by the user. This mechanism is continued iteratively until a `NegotiateLicensesResponse` is received which contains an empty `UnsatisfiedLicenses` list.

4.6.5 The actual getmap Request

Once both license-negotiation mechanisms have cleared, CubeXPLOR finally generates the main web page containing the GetMap requests as HTML `` references. An extra parameter is added to each of the GetMap requests, that being an `ACCEPTED` parameter whose value is the accepted-licenses token for that server. When CubeSERV receives a GetMap request (which is ultimately made by the browser, not CubeXPLOR), it performs a license check and verifies that all required licenses are mentioned in the specified accepted-licenses token. If there are required licenses that aren't mentioned in the accepted-licenses token, then CubeSERV returns an `UnsatisfiedLicensesExist` exception. However, since CubeXPLOR has gone through a license negotiation, this should never happen.

Note that CubeSERV can generally trust the value of the accepted-licenses token, since this token is encoded in a way that that only CubeSERV can generate it and only CubeSERV can decode it. It is an opaque token to all entities outside of CubeSERV. Thus, the only way a valid accepted-licenses token can exist is if it was obtained through the `NegotiateLicenses` mechanism, which can be seen as a legal contract.

It has been suggested that the contents of the accepted-licenses token should be as specified in the "LAT design doc". This would allow servers from different vendors to share a common pool of licenses if configured with the correct private key. However, the first implementation of this mechanism uses a vendor-specific private encoding.

4.6.6 Request Types other than getmap

All of the steps above that deal with GetMap requests also apply to all of the other WMS requests (with the exception of `GetCapabilities`). However, for the first implementation of these mechanisms, it is assumed that only GetMap requests will be under license-management control.

4.7 Notice Acknowledgement Token Specification

The remainder of this document specifies the NAT, a system-independent data structure for representing and sharing notice acknowledgement state information. The NAT provides a simple, extensible representation of an acknowledgement event. For environments where security and tamper-resistance are required, appropriate elements are defined; environments where this is not required need not use or implement these capabilities.

The NAT can be transmitted with a request as the payload of an HTTP cookie or the value of an HTTP extension header. Cooperating web services that follow this specification will be capable of understanding each other's NATs.

4.7.1 The Notice Acknowledgment Token

The format of a NAT is described in this section. NATs are constructed by a server either strictly for its own use or with the intent of sharing state information amongst a set of cooperating servers.

Servers that fulfill part or all of a user's service request by making one or more service requests to other servers ("cascaded operation") are required to forward NATs provided to them by the user or another server. When used in this environment and any environment where servers are loosely associated, an implementation must select NAT attribute types carefully to maximize interoperability.

Note that a server is free to ask a client to delete a NAT (e.g., by setting an expired cookie with the same name) or replace a NAT with a newer instance.

4.7.2 NAT Syntax

A notice acknowledgement token has the following general format:

```
nat = nat-name "=" nat-value
```

Following [RFC 2109](http://www.rfc-editor.org/rfc/rfc2109.txt) (<http://www.rfc-editor.org/rfc/rfc2109.txt> Section 4.3.4), an (unordered) set of NATs is represented as

```
nats = nat *((";" | ",") nat)
```

That is, two or more NATs can be combined for transmission by separating them with a ";" or "," character.

A NAT with an invalid nat-name or nat-value is ignored.

```
nat-value = mime_encode(unsecure-nat) | mime_encode(secure-nat)
```

```
nat-name = token
```

```
unsecure-nat = av-pairs
```

```
secure-nat = hmac hmac-nat
```

```
hmac = "HMAC=<"> hmac-value <">
```

```
hmac-nat = ";" clear-nat ";" enc-method(av-pairs)
```

```
clear-nat = *1("Version=<"> version <"> ";" "Secure=<"> enc-method <">
```

Note:

The secure-nat syntax, which is optional, has been carefully designed to be secure, incur reasonable encoding overhead, and simplify implementation. The secure-nat syntax is used to protect integrity and provide privacy. Integrity is guarded through a cryptographically-secure keyed message authentication code; in addition, attributes may optionally be kept private by encipherment. Note that in the absence of precautions to maintain privacy, inspection of NATs can reveal a portion of a user's access history. In situations where NATs are not transmitted over secure end-to-end connections, such as

those that can be provided by SSL, the secure-nat syntax should be considered. Definitions of encryption and message digest algorithms, and sharing and management of encryption keys are outside the scope of this document.

The `mime_encode` function represents the application of base64 encoding to the given syntactical element (see Section 4.7.3). This encoding prevents elements of the cookie value from conflicting with the syntax of the `Cookie` header or an HTTP message-header and allow binary data to be sent reliably over networks and heterogeneous platforms.

```

av-pairs      = av-pair *("; " av-pair)
av-pair       = attr-name "=" attr-value
attr-name     = token
attr-value    = quoted-string
quoted-string = <"> text-subset <">
text-subset   = <any TEXT except separators>

TEXT          = *<any CHAR except CTLs but including SP>

escaped-char  = "%" HEXDIGIT HEXDIGIT
HEXDIGIT     = "A" | "B" | "C" | "D" | "E" | "F"
              | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT
DIGIT        = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
              "9"
HEXSTRING    = 1*(HEXDIGIT)

token         = 1*<any CHAR except CTLs or separators>
separators   = "(" | ")" | "<" | ">" | "@"
              | "/" | ";" | ":" | "\" | "<">
              | "[" | "]" | "?" | "="
              | "{" | "}" | SP | HT
CTL          = <any US-ASCII control character
              (octets 0 - 31) and DEL (127)>
CHAR         = <any US-ASCII character (octets 0 - 127)>

```

Attributes (`attr-name`) are case-insensitive. One or more spaces are permitted before and/or after a ";" and before and/or after a "=".

An `attr-value` can contain arbitrary OCTETS through the `escaped-char` syntax. A literal "%" character must itself be escaped as the three characters "%25" and a literal ";" character must be escaped as "%3b".

This syntax allows a NAT to appear as the value of the `Cookie` request header or of an HTTP extension header. It is beyond the scope of this document how NATs are passed from system to system, however.

4.7.2.1 NAT Names

It is recommended that NAT names (`nat-name`) begin with the four-character long prefix "NAT-" so that they can be readily recognized. For example, HTTP `Cookie` headers bearing two NATs might use any of the following cookie name syntaxes:

```

Cookie: NAT-METALOGIC=... ; NAT-DSS=...
Cookie: NAT-METALOGIC.COM=... ; NAT-DSS.CA=...

```

Cookie: NAT-METALOGIC.COM-17=...
Cookie: NAT-METALOGIC.COM-WMS-17=...

The goal in selecting a syntax is to provide a way for applications to quickly locate NATs that they generated or are interested in. Since a server may issue multiple NATs as HTTP cookies, each such cookie must have a distinct nat-name. This may be achieved by appending a sequence number, time of day, or hash value, for example.

4.7.2.2 NAT Reserved Attributes

All of the attribute names (attr-name) defined in this section are reserved. All of the attributes are optional, except HMAC and Secure, which are required only when forming a secure-nat.

An implementation may use syntactically valid, unreserved attribute names for its own purposes. Unrecognized and invalid attribute names should be ignored by servers. Attribute names are case insensitive. The relative ordering of attributes is immaterial, except as stated otherwise.

A syntactically invalid nat will be ignored. If duplicate attribute names appear in a nat, a server should treat the nat as invalid and ignore it.

A resource-uri, which is used below, is defined as follows:

```
resource-uri = URI | relative-uri | uri "/"* | relative-uri "/"*
```

1. Attribute Name: CreatorURI (*optional*)
Attribute Value: a URI

This URI identifies the server, application, or jurisdiction that created this NAT. If it is an absolute URI, it also establishes a base URI for relative URIs within this NAT. Examples:

```
CreatorURI="https://fedroot.com:8443"  
CreatorURI="http://fedroot.com/cgi-bin/dacs"  
CreatorURI="MYFED::MYJURISDICTION"
```

2. Attribute Name: BaseURI (*optional*)
Attribute Value: a URI

This absolute URI establishes a base URI for relative URIs within this NAT. If CreatorURI is present, this URI overrides it as the base URI. Examples:

```
BaseURI="https://fedroot.com:8443"  
BaseURI="http://fedroot.com/cgi-bin/dacs"
```

3. Attribute Name: ResourceURIs (*optional*)
Attribute Value: resource-uri *(SP resource-uri)

This attribute identifies one or more resources that have been acknowledged (that is, for which one or more acknowledgements have been received). If resource-uri is an absolute URI, it identifies the resource. If resource-uri is a relative-uri and

neither the BaseURI nor the CreatorURI attribute are present, the implied default base URI is effectively "any server". For example, if no BaseURI and CreatorURI attributes are present, a relative-uri of "/index.html" refers to a resource of that name anywhere. This behaviour can be useful in environments where web content and resource naming are coordinated across servers.

If a resource-uri ends in the two characters "/*", it refers to all URI that are subordinate or equivalent to the absolute or relative URI. Either or both of the characters "/*" may be URL-encoded ("/" by "%2f" and "*" by "%2a") to prevent this interpretation. Refer to Section 4.7.2.3.

It is possible for a particular resource to be known by more than one URI. There is no requirement for two distinct URI to be recognized as referring to the same resource, therefore each may need its own acknowledgement. It is left to each server to decide whether a request URI matches a resource-uri. For example, domain names may not need to match exactly, such as when a web site is mirrored, or some general mapping might be applied to determine whether a request URI matches a resource-uri.

Example:

```
ResourceURIs="https://fedroot.com/index.html"  
  
-- This identifies a single web page.
```

Example:

```
ResourceURIs="/foo.html /bar.html /baz.html"  
  
-- If BaseURI="https://fedroot.com", ResourceURIs identifies  
https://fedroot.com/foo.html, https://fedroot.com/bar.html, and  
https://fedroot.com/baz.html.
```

Example:

```
ResourceURIs="/images/*"  
  
-- Assuming that BaseURI="https://fedroot.com", this  
specifies  
all URI that are subordinate or equivalent to the absolute  
URI  
https://fedroot.com/images, such as  
https://fedroot.com/images/dog.gif,  
https://fedroot.com/images/lines/dotted.gif, and  
https://fedroot.com/images.
```

Example:

```
ResourceURIs="https://fedroot.com/*"  
  
-- This identifies all URI under https://fedroot.com.
```

Example:

```
ResourceURIs="https://fedroot.com/foo/%2a"  
  
-- This specifies exactly one URI, namely  
https://fedroot.com/foo/*.
```

Example:

```
ResourceURIs="https://fedroot.com/foo*"
```

```
-- This specifies exactly one URI, namely
https://fedroot.com/foo*.
```

4. Attribute Name: Version (*optional*)
Attribute Value: "1"

This attribute specifies the version of the specification used for this nat-value. If present, this attribute must appear first. If absent, the attribute defaults to:

```
Version="1"
```

The only attr-value permitted by this version is "1".

5. Attribute Name: Secure (*required for secure-nat*)
Attribute Value: "no" | enc-method

If the value is "no", the av-pairs have not been encrypted and no HMAC attribute is present. A value other than "no" describes the method used to encrypt the following text and the digest algorithm used to compute the value of the HMAC attribute. Its syntax, borrowed from that used by OpenSSL, is:

```
cipher-name *1(cipher-mode) *1(digest-name)
```

Examples of enc-method: aes128-cbc-sha1, aes128-cfb, aes192-cbc, aes256-ofb, des-md5, des3, desx-sha256

6. Attribute Name: HMAC (*required for secure-nat*)
Attribute Value: hmac-value

This is the Keyed-Hash Message Authentication Code (HMAC), represented as a HEXSTRING, computed using the digest algorithm specified or implied by the Secure attribute's value and computed over hmac-nat. SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 are recommended for this purpose. The HMAC is used to authenticate the NAT and protect integrity by detecting tampering.

7. Attribute Name: State (*optional*)
Attribute Value: a URI | local-state-identifier

This attribute identifies server-side state information for this NAT. Rather than building a large NAT, the creator of the NAT may construct a NAT that simply points to state information. There are two types of state information: public and private.

- a. Public state:
If the attr-value is a URI, it identifies a resource that contains additional information (TBD, but presumably it is an XML document). Other cooperating servers may be able to use this URI to obtain attributes described here but not included in the NAT at the discretion of the creating server. (TBD: this service might be used to decrypt and/or validate a NAT associated with it)
- b. Private state:
If the attr-value is not a URI (e.g., it is not prefixed by a recognized

scheme), it identifies information local to the creating server (not available to other servers). This might be a unique database key, for example.

8. Attribute Name: NoticeURIs (*optional*)
Attribute Value: notice-uri *(SP notice-uri)
where notice-uri = a URI | relative-uri

This is a space-separated list of URIs, each of which represents the text of a notice that is associated with ResourceURIs and that has been acknowledged by the user. Each relative-uri is relative to the CreatorURI, which must be provided. To test if a notice-uri matches the URI of a notice associated with a service request, the two URI are compared for equality.

9. Attribute Name: NoticeDigestMethod (*optional*)
Attribute Value: digest-name

This attribute value is the digest algorithm (see the Secure attribute) used to compute the NoticeDigest attribute value elements. If not provided, the digest-name given by the Secure attribute's value must be available and will be used; a weaker but more efficiently computed digest might be sufficient for this purpose, however, such as a 32-bit CRC or MD5.

10. Attribute Name: NoticeDigest (*optional*)
Attribute Value: a space-separated list of HEXSTRINGs

This is a space-separated list of HEXSTRINGs, each corresponding to an element of NoticeURIs, that is the message digest of that NoticeURIs computed using NoticeDigestMethod (or failing that, the algorithm given by the Secure attribute's value). There must be exactly as many elements in this list as are in NoticeURIs, otherwise the NAT is invalid. The purpose of this attribute is to help detect if any NoticeURIs has changed and therefore might need to be acknowledged by the user again.

11. Attribute Name: Expires (*optional*)
Attribute Value: a date string

This is a date field, as specified in the Netscape HTTP Cookie Spec:

wdy, DD-Mon-YYYY HH:MM:SS GMT

While clients are advised of the lifetime of an HTTP cookie at the time it is issued, nothing compels a client to destroy the cookie at that time. An explicit expiry date can be enforced by a server, however, and can be used with mechanisms other than HTTP cookies. The NAT is invalid if this field is incorrectly formatted.

4.7.2.3 URI Matching

For an otherwise valid NAT to potentially be applicable to the URI of a service request, one of the NAT's resource-uri must either be the same as the service request URI or, if a resource-uri ends in "/*", match as an ancestor of the URI. The latter case is called a "wildcard match".

The URI "https://fedroot.com/cgi-bin", for example, is considered to be an ancestor or parent of the URI "https://fedroot.com/cgi-bin/program". The former URI's path component has two elements, the latter has three elements. The one-element URI path "/" is considered to be the ancestor of all other paths.

The "*" operator, which matches zero or more elements, has special meaning only when it appears as a path element at the end of a resource-uri. For instance, the URI path "/cgi-bin/*" is considered to be the ancestor of all paths having the prefix "/cgi-bin/" and matches service requests for "/cgi-bin/printenv", "/cgi-bin/", and "/cgi-bin".

Before matching a service request URI against NATs, the URI is converted into a canonical form. Any trailing "/" characters are stripped off. As a special case, the URI path "/" is unchanged.

The server examines the resource-uri of NATs to find one having the most specific URI path that applies to the service request URI; that is, it conducts a search to find the resource-uri that has the greatest number of components in common with the service request. If no exact match is found, the search will consider increasingly general resource-uri. The resource-uri that matches the URI most closely (i.e., has the greatest number of matching components) determines the applicable NAT, if any.

If two or more resource-uri "tie" (e.g., because of duplicates), one will be chosen arbitrarily.

4.7.3 Encoding for Transport

To ensure that a NAT can be safely transmitted between systems, it is encoded using Base64 Content-Transfer-Encoding, as specified by [RFC 2045](#) (Section 6.8). While not providing additional security, this encoding offers some protection against casual inspection of NAT contents.

4.7.4 Implementation Notes

4.7.4.1 NAT HTTP Header Syntax

A NAT need not be transmitted as the payload of an HTTP cookie, although that will likely be the most common method. Instead, NATs can be transmitted using an HTTP entity-header extension header. The field name "NoticeAcknowledgements" is recommended:

```
NoticeAcknowledgements: NAT-CUBEWERX=..., NAT-DSS=...
```

Like all entity-headers, this name is case-insensitive.

This header may not be repeated unless ", " is used as the separator instead of ";". See [RFC 2616](#) (<http://www.rfc-editor.org/rfc/rfc2616.txt> - Section 4.2).

4.7.4.2 Multiple NATs

When there are multiple NATs, no relative ordering is imposed. In the event that more than one NAT in a list corresponds to the same resource, the resulting behavior is undefined and may depend on the order in which the NATs are processed.

4.7.4.2.1 Resource Name Mapping

It is up to a server whether the acknowledgement of a particular notice for one resource can automatically be applied to another resource. For example, suppose that a user acknowledges notice N_1 upon requesting resource R_1 . Subsequently, the user requests resource R_2 from the server and it happens that notice N_1 also applies to R_2 , although this fact is not recorded in a NAT. It is implementation and context dependent in cases like this whether the server requires the user to acknowledge N_1 again specifically for R_2 or automatically accepts the earlier acknowledgement. Similarly, if a user's NATs collectively indicate that all notices associated with a request have been acknowledged but no single NAT asserts this for the request, the outcome is server-dependent.

4.7.4.2.2 NAT Creation and Merging

Although a NAT may exist that indicates that an acknowledgement has already been obtained, a server may issue a new, more specific NAT or a NAT with a different Expires field. For example, in the example scenario described in the previous section a server might replace the existing NAT for R_1 with one that lists both R_1 and R_2 .

A server may return multiple NATs, adding new ones and deleting or replacing existing ones.

4.7.4.2.3 Case Sensitivity

The path components of two URI are compared case sensitively. The scheme and authority components are compared case insensitively.

4.7.4.2.4 Server Autonomy

Having received a NAT as a result of accessing a given resource, a client should not assume that a later request for the same resource will not involve notice acknowledgements. A server may arbitrarily decide that acknowledgements are needed, a notice may have been modified, a new notice may have been added, or the NAT may have expired.

4.7.4.2.5 Minimal Implementation

A minimal server implementation would issue and recognize NATs that consist simply of a base64 encoded ResourceURIs attribute:

```
"NAT-DSS=" mime_encode("ResourceURIs=" ResourceURIs)
```

4.7.4.2.6 Middleware Support

In support of thick clients and middleware architectures, an implementation might include two useful web services:

- ***NAT creation request***
An authorized client would request that a NAT be created by the server and returned. The request's arguments would describe the content of the NAT and possibly select a format for the returned NAT (HTTP cookie, HTTP header, XML document, and so on)
- ***NAT decoding request***
An authorized client, sending a NAT as an argument to this web service, would receive an indication of whether the NAT was valid and of its contents (e.g., as an XML document).

These hypothetical services are not described further here.

4.7.5 See also

DACS man pages: [dacs_notices\(1\)](#), [dacs.nat\(1\)](#).

4.7.6 Author

Distributed Systems Software (www.dss.ca) and Metalogic Software (fedroot.com).

4.7.6.1 Note

DSS and Metalogic, the authors of this specification, hereby permit anyone to implement this specification without fee or royalties.

4.7.7 Copying

Copyright (c) 2003-2005 Distributed Systems Software. See the [LICENSE](#) file that accompanies the distribution for licensing information.

5 Implementation: UniBW

In the GeoDRM Thread of the OWS3 initiative, the University of the German Armed Forces (UNIBW) addressed the following items:

- WMS with click-through license (client and server component)
- WFS with click-through license (client and server component)

Additionally we worked together with the other participants in this thread for the development of the license model.

Our approach was to reuse existing work as much as possible:

- We based our implementation on existing standards from OASIS: WS-Security – so that specification & implementation effort is saved and resulted OGC services are compatible with web services coming from the IT-world;
- We adopted a proxy approach – so that existing services can be used without modifications

5.1 General approach

Our general approach for GeoDRM is to separate the security and DRM aspects technically from the Geo functionality as much as possible. The advantage of this approach is that existing security concepts and implementations from IT industry, specifically the WS-Security standard and extensions, can be leveraged for implementing security and DRM aspects for geo Web services. The second advantage is the possibility of using geo Web service implementations (e.g., WMS, WFS) without modifications by placing DRM functionalities as separate services between geo Web client and geo Web server.

The basis for this approach is to use the SOAP DCP for all communication where security information is involved (such as identity credentials, signatures, licenses, authorization certificates etc.). The security information is carried in the SOAP header part, as specified in WS-Security and the related standards. The geo data specific requests and answers are carried in the SOAP body and need not be modified in any way. (See figure 7).



Figure 7: Request with Security Information and SOAP

The next step is to separate the *geoprocessing service* from the *security service*. All requests and answers are routed through the *security service*, which processes the security information in the header (usually together with the body information, e.g., to determine from a WMS request which layers are requested and which licenses are needed). The *geoprocessing service* needs no understanding of security aspects and processes only the body information. As a consequence, existing geoprocessing service implementations can be used without modifications in GeoDRM scenarios. By using a protocol converter from SOAP to HTTP get/post, it is even possible to use geoprocessing services not supporting the SOAP DCP, since the additional security information needs not to be communicated to the *geoprocessing service*(see figure 8).

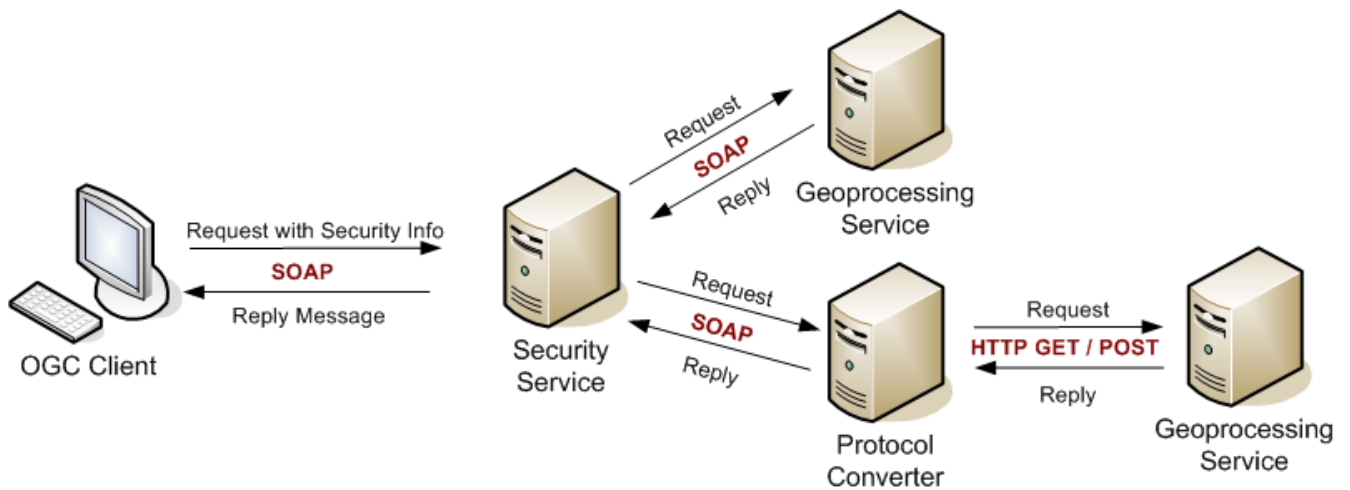


Figure 8: Request with SOAP and HTTP GET protocols

5.2 Implemented Use-Cases

UniBW prototypically implemented the following use-cases (see chapter where use-cases are presented):

- Anonymous User
- Named User

5.3 The Agreement Workflow

Agreeing with the terms of use happens by some out-of-band mechanism. We implemented this as an HTTP interface. The workflow is as follows:

1. (Named User) The user registers on some website and receives some credentials. In our implementation we used user name and password, but other mechanisms such as SAML or PKI could be used as well.

2. (Anonymous User) Alternatively the client software makes a `getSession` request in order to receive the credentials (in our implementation a user name and a password to be used for the length of the session)
3. The user can visit the website anytime to agree with different licenses
4. The server can change the licenses anytime. In this case the user should revisit the website and re-agree to the license
5. When a user makes a request to the server, the user should send his credentials together with his request. Based on these credentials, the server would decide if the request is allowed or refused.
6. If a request is refused, the server returns an exception message to the client. The exception includes the URL of the website that the user should visit in order to register / agree to the terms of use.

The following figure 9 illustrates the agreement workflow in the two use-cases (named user / anonymous user):

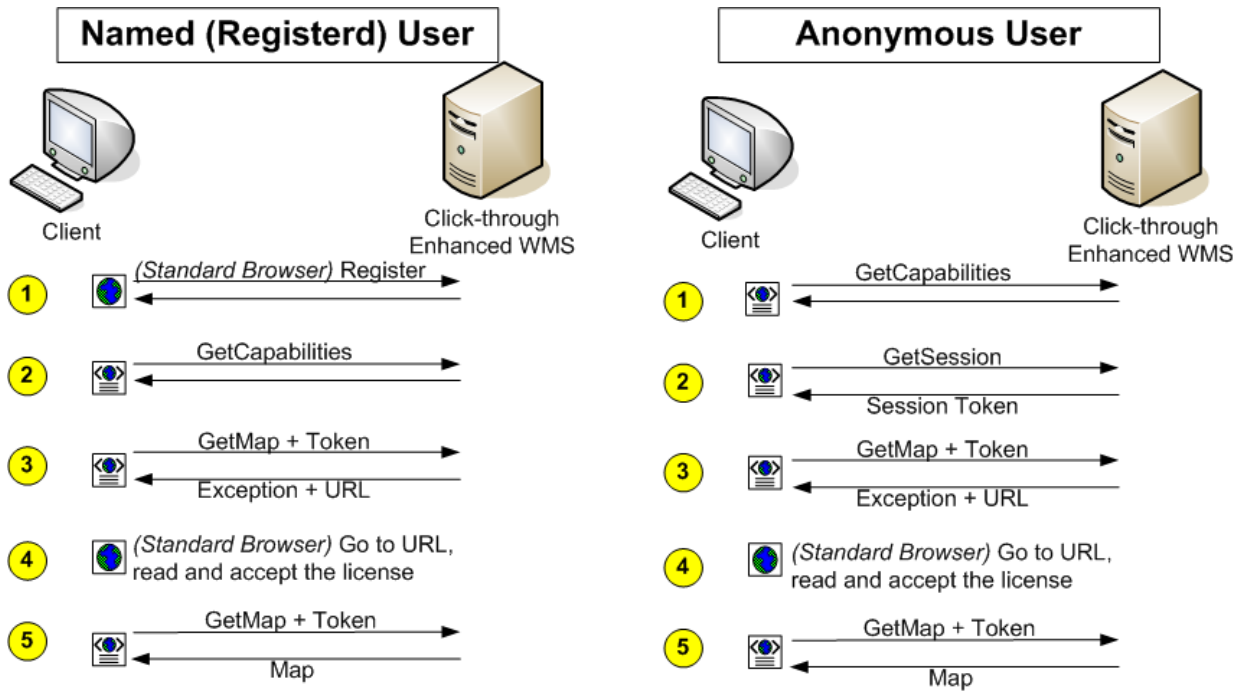


Figure 9: Request with SOAP and HTTP GET protocols

5.4 Click-through licensing for Named Users

Named users are frequent users that only want to acknowledge the terms of use once. The users will first register with the service and then authenticate themselves when using the service.

We consider the registration process as not subject to specification; it can be implemented in various ways (for example as a web site). After being successfully registered users will receive some form of credentials (as for example a user name and a password) that will be presented to the service for the purpose of authentication.

The credentials shall be sent to the server together with the every request, encoded in the SOAP header as described by the WS-Security³ standard from OASIS⁴.

5.4.1 WS-Security and token profiles

WS-Security is a standard from OASIS as of March 2004. The specification “describes enhancements to SOAP messaging to provide message integrity and confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies. This specification also provides a general-purpose mechanism for associating security tokens with message content.”

Currently the following four token profiles are defined and approved by OASIS:

- Username Token Profile
- X.509 Token Profile
- SAML Token Profile
- REL Token Profile

A token profile for Kerberos is in work within the WSS TC at OASIS.

5.4.2 Access Control for OGC SOAP messages

Access Control is done by encoding WS-Security Tokens in the header of the SOAP message.

Remark:

- Because of the requirements of the OWS3 project we (UniBW) consider username / password authentication as described in the Username Token Profile of WS-Security⁵ to be sufficient. Nevertheless, as mentioned above, WS-Security also supports other kinds of authentication. We believe that as long as all participants respect the WS-Security specification the implementations will be compatible and interoperability will be achieved.

An example of a WMS `getMap` request using SOAP is presented below. The header of the SOAP message contains a username token with hashed password, nonce and creation date. The nonce and creation date help preventing reply attacks. The SOAP message is encoded as described in the OGC document 04-050r1 “WMS Change Request: Support for WSDL & SOAP (WMS-WSDL)” / OGC document 02-017r1 “WMS Part 2: XML for Requests using HTTP Post (WMS POST)”.

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
```

³ For more information see: <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

⁴ <http://www.oasis-open.org>

⁵ <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>

```

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
  <wsse:Security SOAP-ENV:mustUnderstand="0"
    xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <wsse:UsernameToken>
      <wsse:Username>cristian.opincaru</wsse:Username>
      <wsse:Password
        Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordDigest">
        McUxzsc9q09XRdPwMc5nFmWVJFQ=
      </wsse:Password>
      <wsse:Nonce>p2bbLkMtW5rLiYqzRaA3BA==</wsse:Nonce>
      <wsu:Created
        xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
        2005-09-05T14:15:52.522Z
      </wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <wms:GetMap service="WMS" version="1.1.0"
    xmlns:wms="http://www.opengis.net/wms">
    <sld:StyledLayerDescriptor
      xmlns:sld="http://www.opengis.net/sld">
      <sld:NamedLayer>
        <sld:Name>Gebaeude</sld:Name>
        <sld:NamedStyle>Geb</sld:NamedStyle>
      </sld:NamedLayer>
      <sld:NamedLayer>
        <sld:Name>Hausnummer</sld:Name>
        <sld:NamedStyle>Hn</sld:NamedStyle>
      </sld:NamedLayer>
    </sld:StyledLayerDescriptor>
    <gml:BoundingBox srsName="EPSG:31467"
      xmlns:gml="http://www.opengis.net/gml">
      <gml:coordinates>
3476891.0179,5367516.2000,3477006.5821,5367588.2000
      </gml:coordinates>
    </gml:BoundingBox>
    <wms:Output>
      <wms:Format>image/png</wms:Format>
      <wms:Transparent>TRUE</wms:Transparent>
      <wms:BGcolor>0xFFFFFFFF</wms:BGcolor>
      <wms:Size>
        <wms:Width>594</wms:Width>
        <wms:Height>370</wms:Height>
      </wms:Size>

```

```

        </wms:Output>
    </wms:GetMap>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

A similar example for a WFS `getFeature` request is provided below. Since the mechanisms reside in the transport protocol (they are not specific to any OGC specification), the SOAP messages are similar.

```

<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <wsse:Security SOAP-ENV:mustUnderstand="0"
      xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>crisian.opincaru</wsse:Username>
        <wsse:Password
          Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordDigest">
          +24f7jdQ8+VtTFCX+xCWJQhFAFc=
        </wsse:Password>
        <wsse:Nonce>nU2Jaiy6iwqjBJ3aiO2FtA==</wsse:Nonce>
        <wsu:Created
          xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
          2005-09-06T15:12:55.750Z
        </wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wfs:GetFeature outputFormat="GML2" request="GetFeature"
      service="WFS" version="1.0.0"
      xsi:schemaLocation="http://www.opengis.net/wfs"
      xmlns:gml="http://www.opengis.net/gml"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:wfs="http://www.opengis.net/wfs"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <wfs:Query typeName="Kanal">
        <ogc:Filter>
          <ogc:FeatureId fid="Kanal.43968" />
        </ogc:Filter>
      </wfs:Query>
    </wfs:GetFeature>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

5.5 Click-through licensing for named users

In order to facilitate click-through licensing for anonymous users, a new operation should be introduced: `getSession`. The operation should be called by a client accessing a GeoDRM enabled WMS / WFS service after the client has executed the `getCapabilities` request. `getSession` will return a credential / token that the client should use in his subsequent requests to the OGC service. The token is used for session management.

Remark:

- This operation is optional for services / clients to implement. As seen in the previous chapter, click-through licensing can also be accomplished by using named users. Furthermore, other mechanisms such as cookies can be used instead of this operation in some cases (for example if the client is a browser).

5.5.1 GetSession Definition

Request Parameter	Required/ Optional	Description
SERVICE=WMS/WFS/	R	Service type
VERSION=version	O	Request version
TOKENTYPE=TokenTypeID	R	The type of token to be used. For the moment we proposed <code>UserNameToken</code> that will return a unique user name token. Other tokens such as SAML tokens, cookies, etc. can also be added.

Table 2: GetSession Definition

5.5.2 Example (SOAP)

- The following shows a request to a WMS service for a `UserNameToken`:

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <wms:GetSession
```

```

        service="WMS"
        version="1.3.0"
        tokenType="UserNameToken"
        xmlns:wms="http://www.opengis.net/wms" />
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

- The following shows a possible response to the previous getSession request:

```

<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ows3:UserNameToken
      xmlns:ows3="http://iis.unibw-muenchen.de/ows3">
      <ows3:UserName>sjgeahed102g2w</ows3:UserName>
      <ows3:Password>7263db23jhb</ows3:Password>
    </ows3:UserNameToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

- From this point on the client would use the username / password tokens it received in his subsequent requests to the WMS server (getMap, getFeatureInfo, etc.)

5.5.3 getSession definition in capabilities documents

The getSession method should be advertised in service capabilities documents in the same way the other service operations are advertised. A possible example is shown below:

```

<GetSession>
  <TokenType>UserNameToken</TokenType>
  <DCPType>
    <HTTP>
      <SOAP>
        <OnlineResource
          xlink:href="http://iisdemo.informatik.unibw-
muenchen.de/ows3"
          xlink:type="simple"
        xmlns:xlink="http://www.w3.org/1999/xlink" />
      </SOAP>
    </HTTP>
  </DCPType>
</GetSession>

```

5.6 Service Exceptions

Two new exception codes were introduced in order to signal the client that the authentication was unsuccessful and that he needs to first read terms and conditions. These exceptions are described in the following sections. WMS / WFS services must support these exception codes.

In the exception body, an XML document is sent that contains a link (as xlink) to a website that the client should visit in order to obtain access to the WMS / WFS service. The XML is embedded as CDATA because the WMS 1.3 Specification mandates it.

5.7 Disclaimer not agreed

An OGC service shall respond with a “Disclaimer not agreed” exception if the user made a `getMap` / `getFeature` request containing layers / feature classes for which he did not previously acknowledged the terms of use.

The following shows a SOAP response to a request where the user had not previously agreed to the terms of use:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:ServiceExceptionReport version="1.1.0"
      xmlns:ns1="http://www.opengis.net/wms">
      <ns1:ServiceException code="disclaimerNotAgreed">
        <![CDATA[<OnlineResource
xmlns:xlink="http://www3.w3.org/1999/xlink" xlink:type="simple"
xlink:href="http://137.193.63.168/ows3demo/jsp/main.jsp"
/><Explanation>de.ubm.ows3.server.DisclaimerNotAgreedException:
Gebaeude</Explanation>]]>
        </ns1:ServiceException>
      </ns1:ServiceExceptionReport>
    </soapenv:Body>
  </soapenv:Envelope>
```

5.8 Invalid Credentials

An OGC service shall respond with an “Invalid Credentials” exception if the credentials supplied by the client were invalid (for example the username and password are not correct).

The following shows a SOAP response to a request where invalid credentials were supplied:

```

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:ServiceExceptionReport version="1.1.0"
      xmlns:ns1="http://www.opengis.net/wms">
      <ns1:ServiceException code="invalidCredentials">
        <![CDATA[<OnlineResource
xmlns:xlink="http://www3.w3.org/1999/xlink" xlink:type="simple"
xlink:href="http://137.193.63.168/ows3demo/jsp/main.jsp"
/><Explanation>de.ubm.ows3.server.InvalidCredentialsException:
org.apache.ws.security.WSSecurityException: The security token
could not be authenticated or authorized</Explanation>]]>
        </ns1:ServiceException>
      </ns1:ServiceExceptionReport>
    </soapenv:Body>
  </soapenv:Envelope>

```

5.9 Service chaining: FPS / Cascading WMS

In the case of service chaining, the cascading service should forward the credentials to the cascaded service. For example a FPS would extract the security tokens from the message request and include them in the request that it makes to the WFS (see figure 10).

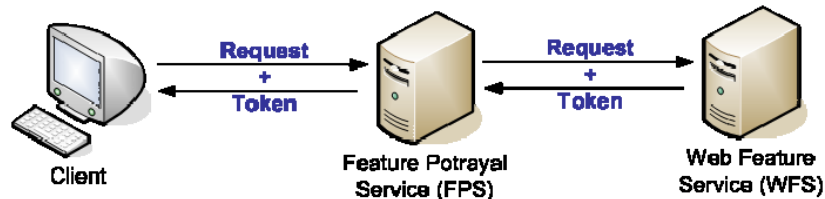


Figure 10: Chaining

If exceptions occur, the cascading service (the FPS in the picture above) shall send them to the client.

Remarks:

1. The trust relation is established between the client making the request and the data provider (the WFS, in the above example). This means that the terms of use are **managed** and **enforced**, by the data provider.
2. This mechanism will not work with some WSS token profiles, for example if PKI is used and the profile mandates that the SOAP body be signed: in this case the cascading service does not have the private key, therefore it will not be able to produce requests signed with the client's digital signature.

3. As the WSS specification allows for more than one security token to be included in the SOAP message, scenarios where a cascading service cascades several cascaded services (see picture below) can be imagined. In this case, the client would include all necessary tokens in his request. The cascading service would forward these tokens (only some of them or all of them), when making its requests (see figure 11).

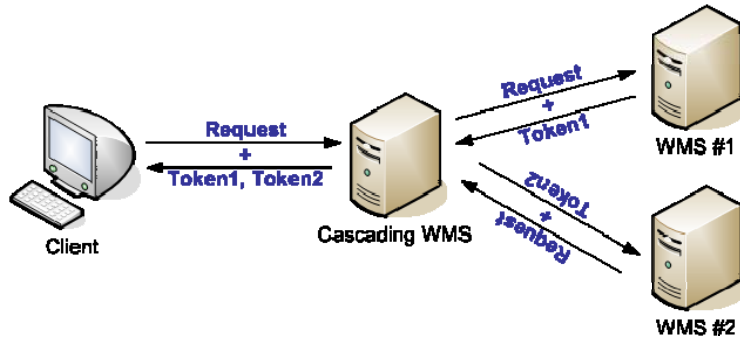


Figure 11: Cascading

5.10 Software implementation description

5.10.1 Architecture

An overview of the ideas behind the chosen architecture can be found in section 5.1 - “General approach“. The following picture shows a typical deployment for a click-through WMS (see figure 12.)

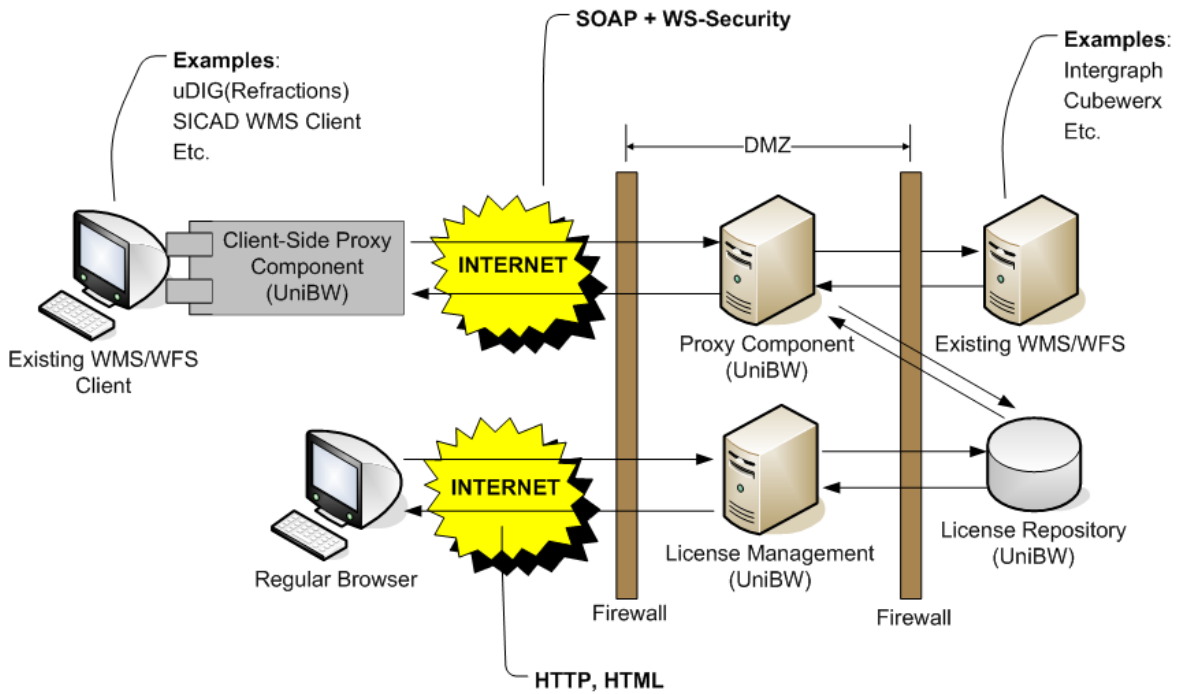


Figure 12: Architecture Implementation UniBW

5.10.2 Components

Server-Side Proxy Component

Description: This component is the façade of the WMS/WFS to the outside world. In a typical deployment this component would be placed in a Demilitarized Zone (DMZ), while the real WMS/WFS service together with the other components would be placed in a protected zone.

This component acts as a proxy for the real service (WMS/WFS). It therefore exposes the same interfaces as the latter one (i.e. getCapabilities, getMap, getFeature, etc.). A client would not see (and would not be aware of) the services behind the proxy. Every request he makes will be sent to the proxy.

Additionally this component converts between SOAP and HTTP GET/POST.

The proxy takes the SOAP request, does the license verification by looking up the credentials in the license repository, and:

- Forwards the request to the real WMS/WFS service (if the client needs not agree to any license)
- Sends an exception back if the client is required to acknowledge a click-through license

Implementation: The component is implemented in Java 1.5 as Apache AXIS web service.

License Management Component

Description: The license management component is responsible for managing the click-through licenses for each user.

Implementation: This component is implemented using JSP (Java Server Pages). The pages are deployed in a standard JSP container (like Apache Tomcat).

License Repository

Description: The license repository stores the click-through licenses, user profiles and session information.

Implementation: MySQL database

Client-Side Proxy Component

Description: This component is the façade of the WMS/WFS to the local client. It is implemented as an applet that is loaded when a web page is opened.

This component receives a standard OGC request and (1) converts it to a SOAP message and (2) injects the credentials in the header of this request.

When receiving the replay, it (1) converts the reply to a HTTP reply and (2) checks to see if there are exceptions in the message. If an exception is found that contains an URL, then this page will be opened in a browser window.

Implementation: The component is implemented in Java 1.5 as an applet.

5.10.3 Licensing of software

All the software developed by us during the OWS3 initiative are licensed using a **dual licensing schema** (*Open Source + Commercial*) similar with the license used for MySQL. The *Open Source* license allows for use, modification and redistribution with the condition that the source is made available, while the *Commercial* license allows companies to include the software in their products without being forced to publish the code.

There is currently no CVS repository set up. To obtain a copy of the source codes send an email at: Cristian.Opincaru@unibw.de.

5.10.4 Tests and Demonstrations

A demonstration of the project can be found at the following URL:

<http://iisdemo.informatik.unibw-muenchen.de/ows3demo/>

6 Implementation: Lat-Ion

An OWS which is able to present a license agreement to a user should extend its capabilities to announce:

- that it is DRM-enabled
- which access control methods it accepts (optional if just anonymous user access is supported)
- access information for the operations described below

A DRM module, no matter whether implemented within an OWS or as a façade to it, should offer two interfaces, as described below.

6.1 GetLicenses operation

The GetLicenses operation allows a client to query the service about the terms of use of given feature types. The response to a GetLicenses operation signals whether the feature types are under terms of use. The diagram below depicts the interaction between client and click-through module (see figure 13).

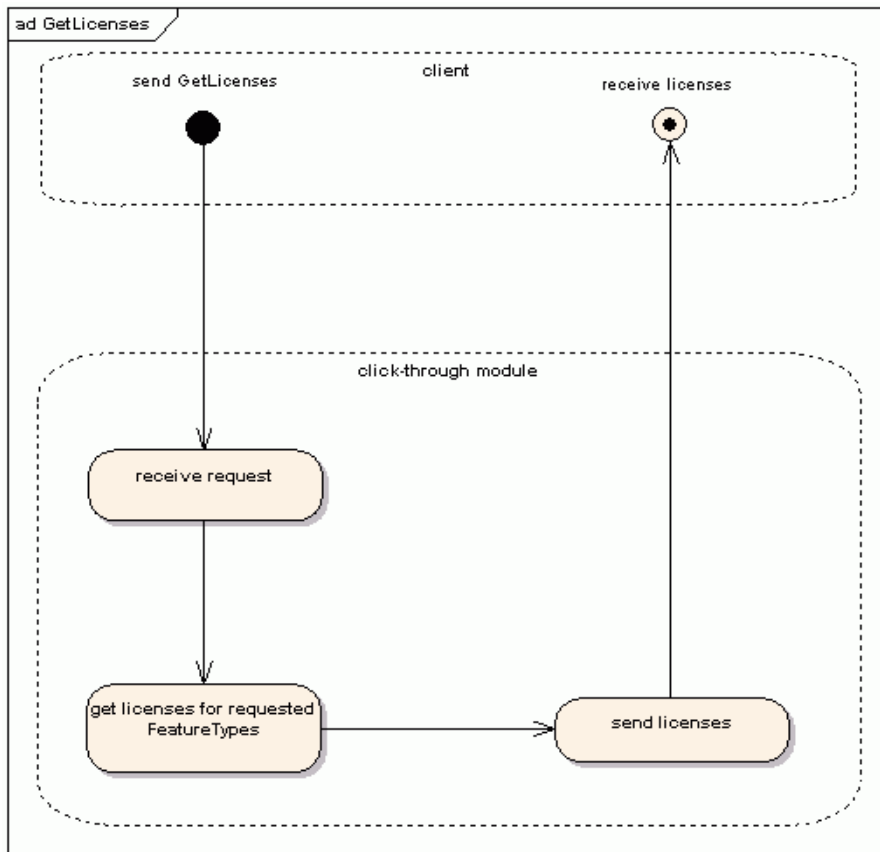


Figure 13: Activity diagram showing a GetLicenses request and response

The parameters of a GetLicenses request are described in more detail below.

6.1.1 GetLicences request

A GetLicenses request allows a client to retrieve license information for a defined service operation on the passed entities. The parameters of this request are shown in the table below.

Request Parameter	Required/Optional	Description
REQUEST=GetLicenses	R	Request name
OWSREQUEST=owsRequestName	R	The name of the request for a defined OWS operation
ENTITIES=entities_list	R	The comma-separated list of entities which are under license for a given OWS request

Table 3: parameters of a GetLicenses Request

6.1.2 GetLicenses request example

Example of a GetLicenses request for an entity (feature type)

<http://some.server.com/ows3/deegreevfs?>

```
request=GetLicenses&  
owsrequest=GetFeature&  
entities=os_poi
```

Here a GetLicenses request is performed for an entity (feature type) which is under a license. As response the DRM module should return the license(s) a user has to agree with if he/she wants to proceed with the GetFeature request.

6.1.3 GetLicences response

The GetLicenses response lists the licences associated with a given entity. This request has an informative character. That is, it has no side effects.

6.1.4 GetLicenses response examples

Example of an GetLicenses response for an entity (feature type) under license:

```
<LicenseInformation xmlns="http://www.deegree.org/security/license">
  <Resource>
    <ServiceType>OGC:WFS</ServiceType>
    <OnlineResource
      xlink:href="http://some.server.com/deegree/wfs"
      xlink:type="simple"
      xmlns:xlink="http://www.w3.org/1999/xlink">
    </OnlineResource>
    <Request>GetFeature</Request>
  </Resource>
  <License id="License1">
    <AssignedObject name="os_poi"
      title="Osnabrueck Points of Interest"/>
    <Conditions>
      A license you can accept or reject...
    </Conditions>
  </License>
</LicenseInformation>
```

The <Resource> element contains information about the service (type) and the operation being queried, defined within the <Request> element. The <License> element has an attribute, id, to identify the license, an <AssignedObject> element describing the feature type (name) and a <Conditions> element. The latter one contains the actual license text.

For a feature type without a licence, the GetLicenses request returns a similar document, but emptied of any <License> elements.

6.1.5 Discussion

Here the approach has been to associate licenses with feature types. No further constraints are taken into account. One could include, for example, parameters for further constraints. This would allow the service not only to restrict access to an entity and an operation but also to an entity and an operation on, say, a given bounding box. Another example might be to restrict the access to an entity (layer) with an operation (GetMap) and a constraint (scale). This more generic approach would extend the above operation to include the optional parameter „constraints“. In this way, licenses can be requested for an operation, e.g. „GetMap or „GetFeatureInfo“, on an entity, e.g. Layer „land_use“, with the constraint „{BoundingBox=1.2,3.4,5.6,7.8;SRS=EPSG:4326}“. This approach allows the service to put a resource under a license, for example, to a specific region and spatial reference system. Though possible, this approach has been rejected during OWS-3 due to the simpler characteristics of the click-through use-case.

The usage of XML-encoded license information instead of HTML pages is useful because not every client is a HTML application running in a browser. Using an XML document enables a

client to process the license document to present it in a way, which best suits the purpose (see figure 14).

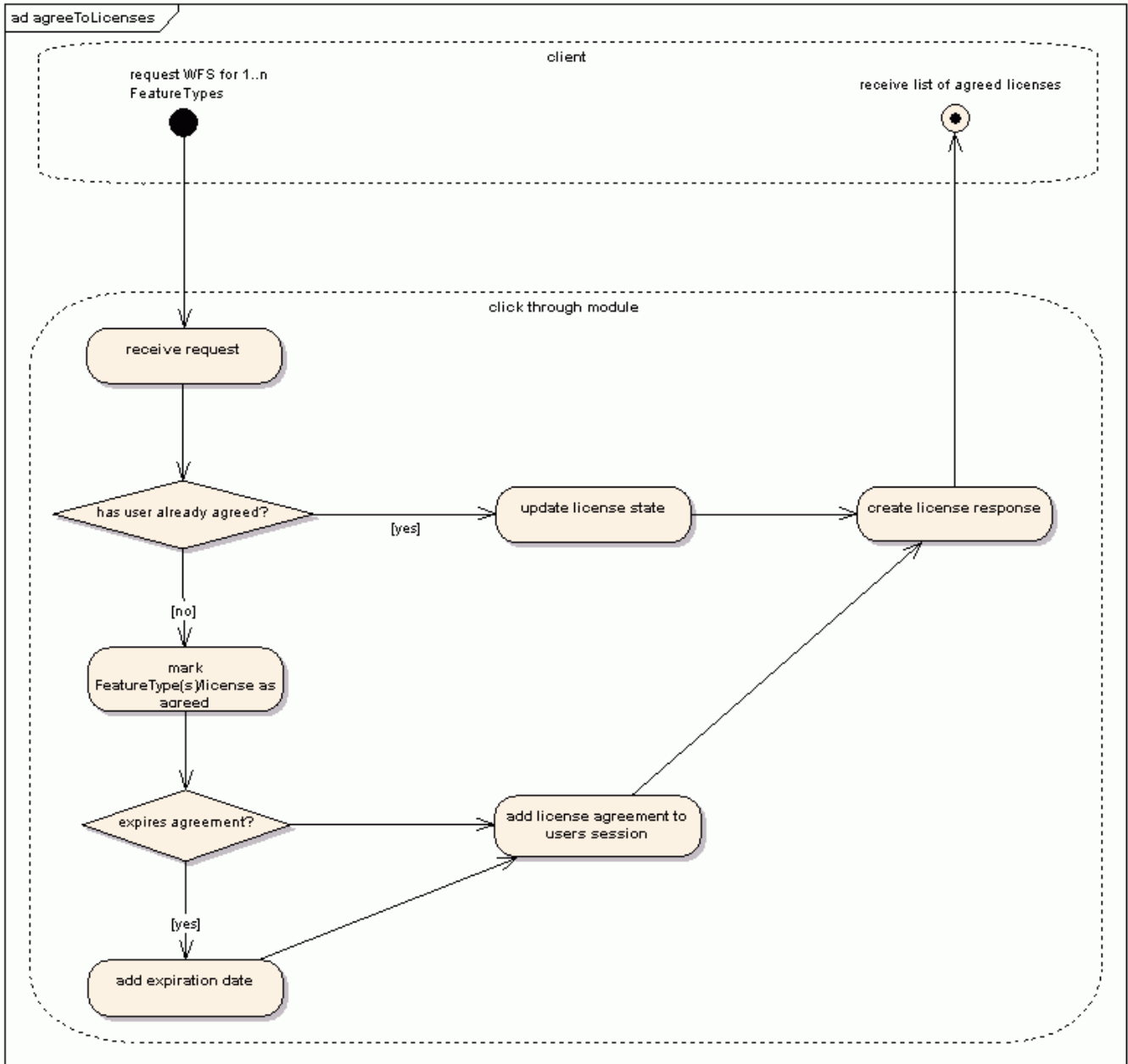


Figure 14: Diagrammatic depiction of a NegotiateTerms request (formerly known as “doLicenseAgreement“ or “agreeToLicense”)

6.2 NegotiateTerms operation

A NegotiateTerms request should be performed if a user agrees to one or more licenses used by a DRM-module. (Please note that this request was formerly known as “doLicenseAgreement” or “agreeToLicense”). The workflow is shown below.

The parameters of this request are described as follows.

6.2.1 NegotiateTerms request

The parameters of a NegotiateTerms request are listed in the table below.

Request Parameter	Required/Optional	Description
REQUEST= NegotiateTerms	R	Request name
LICENSEIDS=list_of_licenses_identities	R	Comma separated list of licenses IDs

Table 4: NegotiateTerms request

6.2.2 NegotiateTerms request example

Example of a NegotiateTerms request for an entity (feature type) under license:

```
http://some.server.com/ows3/deegree/wfs?  
  REQUEST=NegotiateTerms&  
  LICENSEIDS =License1
```

The request above means the user (sending the request) is agreeing with the licenses defined in LICENSEIDS. The request returns the text of the accepted licenses. The accepted licenses are defined by the licensesIDs parameter. Note that this request has the side effect of releasing any entities that were under the license given by the parameter LICENSEIDS.

6.2.3 NegotiateTerms response

The NegotiateTerms response equals that of the GetLicenses response.

6.2.4 NegotiateTerms response example

Listing 4: Example of the response of a NegotiateTerms request for a given license

```
<LicenseInformation xmlns="http://www.deegree.org/security/license">  
  <Resource>  
    <ServiceType>OGC:WFS</ServiceType>  
    <OnlineResource  
xlink:href="http://some.server.com/deegree/wfs/wfs"  
      xlink:type="simple"
```

```

xmlns:xlink="http://www.w3.org/1999/xlink"></OnlineResource>
  <Request>GetFeature</Request>
</Resource>
<License id="License1">
  <AssignedObject name="os_poi" title="Osnabrueck Points of
Interest"/>
  <Conditions>
    A license you can accept or reject...
  </Conditions>
</License>
</LicenseInformation>

```

6.2.5 DRM WFS response without license acceptance

Upon receiving a GetFeature request on a licensed feature type, without previous license acceptance a GeoDRM WFS returns the following service exception:

```

<?xml version="1.0" ?>
  <ExceptionReport version="1.1.0"
    xmlns="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="owsExceptionReport.xsd">
    <ExceptionText>
      User does not have a valid License to access the requested
feature type
    </ExceptionText>
  </ExceptionReport>

```

6.2.6 Discussion

The NegotiateTerms response equals that of a GetLicenses request. The difference is that the NegotiateTerms response basically represents the acceptance of the terms of use. Having performed a NegotiateTerms operation, the user is free to proceed with a WFS GetFeature request. In case he/she has not agreed with any license, i.e. has not performed a NegotiateTerms operation, a subsequent GetFeature request yields a service exception as described above.

To allow a client a variable implementation of license management and agreement, license agreement should not be bundled with OWS requests. (Note that, if a user has not agreed with the licenses for a GetFeature request before performing it the server could return an exception containing a URL pointing to license information as described above instead of a feature collection. This is, however a so-to-speak short cut that has not been implemented.)

It should be recognised that the service should be free to expire the license and thus reject further GetFeature requests. This situation, i.e. expiring a license, is equivalent to putting the feature type under a new license.

6.3 A client application demo

As a demo, a click-through wizard has been implemented in deeJUMP, a degree-based extension of the Java Unified Mapping Platform (<http://www.jump-project.org/>). The click-through wizard is implemented as a JUMP plug-in and is available for download as a JUMP extension. The purpose of the client is to demonstrate the service in user-environment and to contribute to code for other clients used in the OWS-3 initiative. See resource list below.

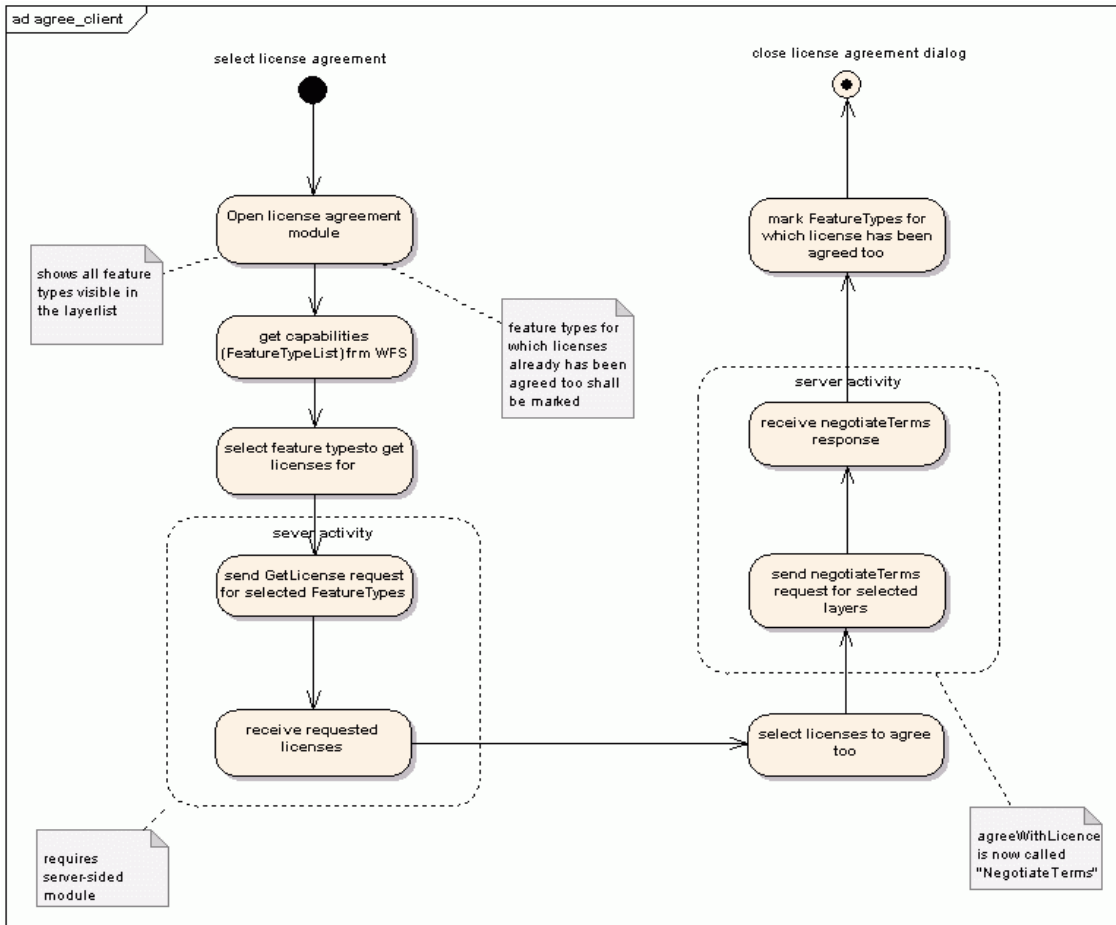


Figure 15: The click-through workflow

For simplicity's sake, the demo application performs NegotiateTerms on a feature type basis. The GeoDRM-enabled demo WFS has two feature types: 'os_poi' (Points of Interest of the City of Osnabrueck) and 'os_free_poi' (same data set, but free as in 'under no licenses').

6.3.1 Click-through on a free feature type

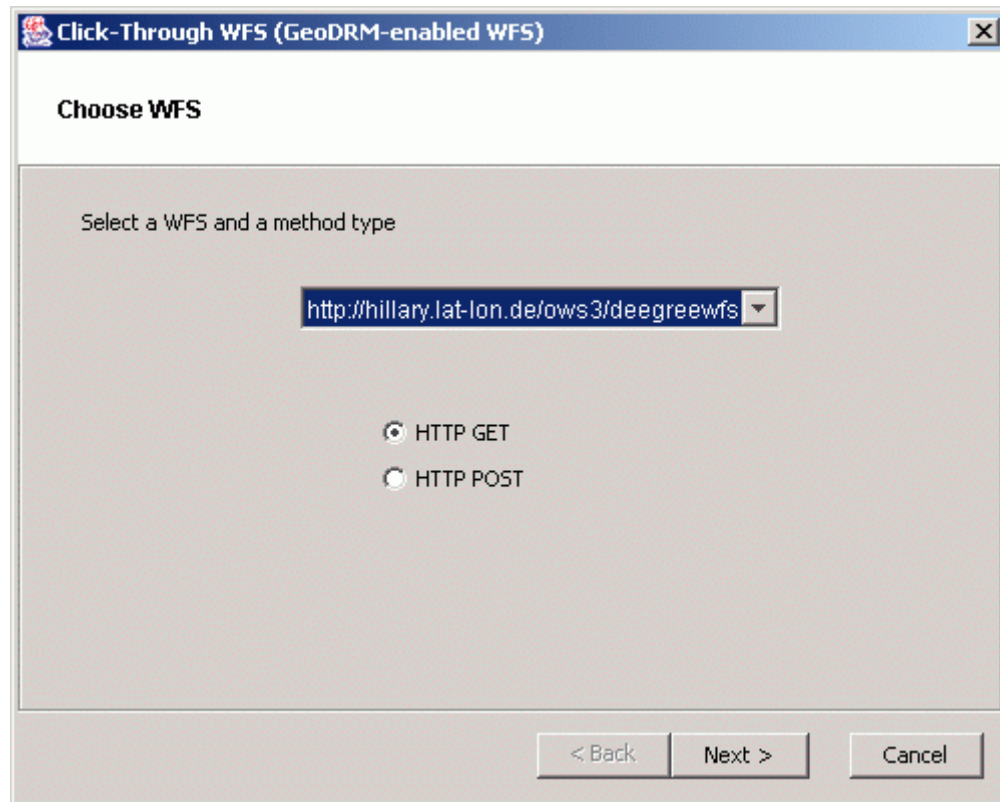


Figure 16: The demo client starts with a dialog allowing the user to select WFS.

The user proceeds to choose a feature type. Having chosen one that is not under licence, e.g. 'os_free_poi', the user may proceed to perform a GetFeature. The fact that no licence is available is recognized by the client, by performing a GetLicences on the entity 'os_free_poi' and with the operation 'GetFeature'. The client changes the wizard workflow to avoid the last step, namely the acceptance/rejection of the licence. (A NegotiateTerms would have no effect whatsoever for a "free" feature type, anyway).

The GUI shows a "Finish" button rather than a "Next" one. For illustration's sake, two text areas show the license info and the WFS Capabilities, respectively.

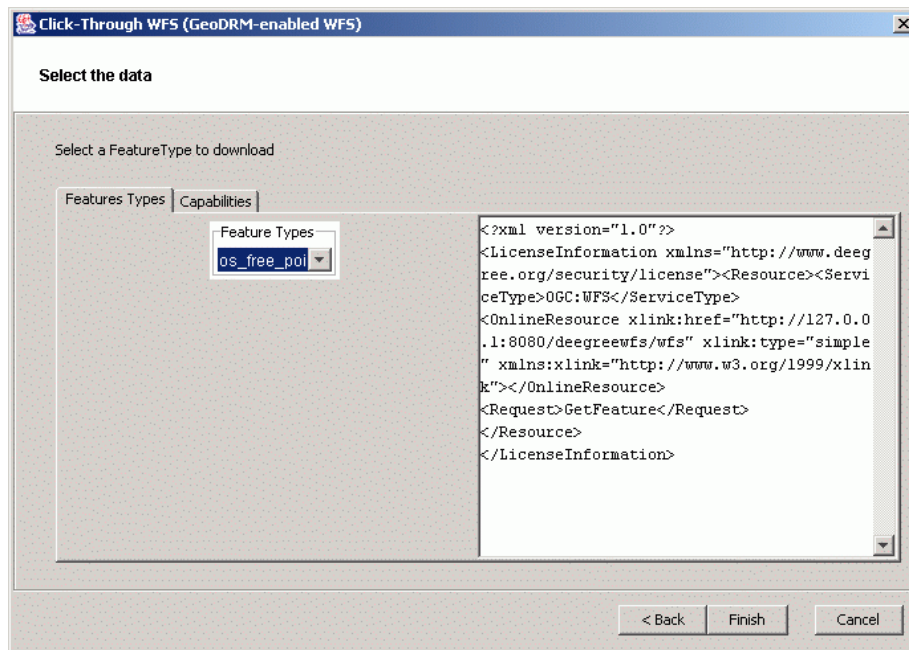


Figure 17: Selection of a feature type, which is under no license

6.3.2 Click-through on a non-free Feature Type

The demo client starts with a dialog allowing the user to select a WFS, as before. The user then proceeds to choose a feature type that is under license/terms of use, 'os_poi'. A 'GetLicenses' is performed, the client is now aware that this FeatureType is under a licence and forces the user to read the terms of use.

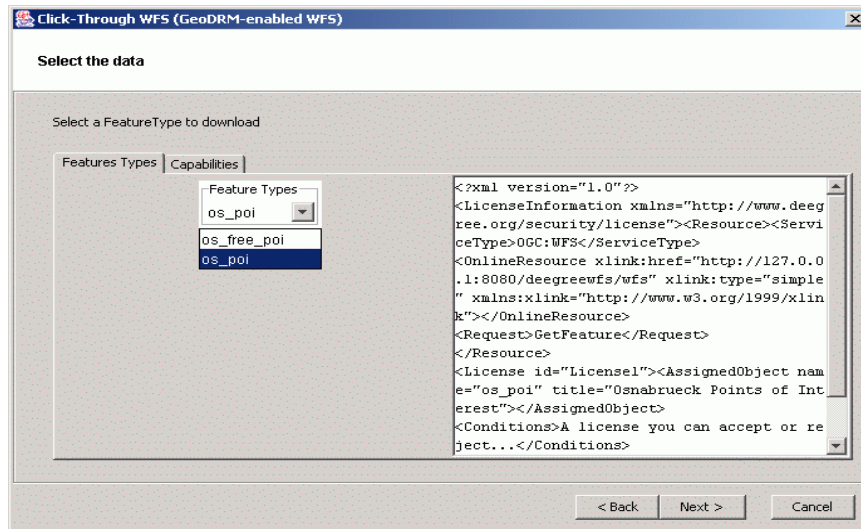


Figure 18: Selection of a feature type, which is under no license

Clicking on "Accept" and then on "Finish" will perform a NegotiateTerms for the license of the chosen feature type. The client then performs a GetFeature request, which returns the expected data. If the user rejects the license, no NegotiateTerms is performed. The service throws an exception as shown in listing 5.

The information about whether the user has agreed with a license or not is held in the session. The adoption of such mechanism is justified by the following: there are services whose task is to perform access control and register user information (see [2], [3] and [4]). As far as sustainability is concerned, it's not feasible to devise a mechanism to persist user information that is not standardized. Though such user-related information could be held in a cookie, which is in some ways a de facto standard, there's still the problem of agreeing on the cookie format.

6.4 Consequences in regard to OGC specifications

The integration of the herein proposed solution for authenticated access to OWS like WFS, WMS, or WCS into the OGC bookshelf of specifications can be achieved in at least three different ways:

- Specify a DRM service as an OGC service on its own right,
- Include DRM operations individually into each service specification, or
- Extend OWS Common by optional DRM operations.

The proposed solution is generic enough to be used as a starting point for each scenario. From an implementer's point of view a more general solution – like the DRM Service or the OWS Common Extension – is preferred. More specifically it is suggested to extend OWS Common by optional DRM operations, since this would lower the implementation barrier – both on service and client side: Neither a completely new service product would need to be implemented, nor would client implementations need to be extended by a new service interface.

If one decides to include the DRM operations proposed here, the WFS specification will need to be changed accordingly. The same applies to other services implementing DRM operations.

6.4.1 OWS Common Change Request

Include GetLicense and NegotiateTerms as optional operations into the OWS Common Implementation Specification.

6.4.2 GML Change Request

The DRM WFS proposed here has no impact on the current GML (3.1) specification.
Interpretation of common elements observed from the disparate implementations.

7 General GeoDRM Capabilities and Term-of-Use Models

The aim of the OWS3.geoDRM thread was the development of a click-through (license) service, which delivers unstructured (legal) text information. Beyond the single text field some more elements were expected to describe the overall processing. Therefore another work package was defined to examine the Terms-of-Use information model more in detail. Some requirements can be derived directly through the given use case in chapter 2. Some requirements are the result of the overall discussion. This chapter describes the relevant requirements and a first modelling approach.

7.1 Requirements

The requirements of paragraph 3.5 "Fully-informed and Trail & Error approach" are relevant for the Terms-of-Use model. The fully-informed process needs a document, which contains all legal information. This document needs to be accessible without any legally binding processing.

The management of Terms-of-Use may be harmonized with-in an SDI community. Therefore a neutral player publishes common Terms-of-Use documents. Because of different products and business relationships different kinds of Terms-of-Use documents are required. Therefore each kind or later called "category" should have a unique identifier and a human readable name. The unique identifier should also be used in the case of multiple instances to reduce repetition.

The Terms-of-Use content may consider WMS layer, GML 2.0 feature, GML 3.0 feature, and more general: "data sets" expressions, complete services with its content or even service farms, with a large variety. An example text could cover "disclaimers for all available services". Therefore the business term "product" is more suitable to describe the resource. It offers a more general term to reduce complexity and to re-use information and processing. The model should offer grouping methods for the "product" level, including hierarchies in analogy of the WMS <layer> element. It should also offer extensive wild-card methods.

The realization of the given use cases will require different business functions. The identified functions in paragraph 3.4 (Access Control) need process descriptions. Because the function "authentication" can be used also in other business relationships without Terms-of-Use acknowledgement, it is useful to encapsulate each business function. Upcoming business functions, e.g. pricing & ordering have similar requirements and should be considered as a placeholder to proof a more general concept already in the design phase.

Because some business models may only require some selected business functions all functions should be described independently. The order of business functions in the workflow is relevant.

The general requirement "Backwards compatibility" was already identified in 3.8. It is full relevant for the information model. A backwards compatible approach is a crucial

design requirement and should be supported in the Terms-of-Use information model. The OGC OWS Common “getcapabilities” operation and “capabilities document” should be supported.

7.2 Schema Overview

The explained requirements could be solved and resulted in a Terms-of-Use model schema with a number of 37 elements. The W3C XML Schema language was used to express the model. Some elements were re-used from OGC OWS common⁶. These elements were not re-described in this document. Both XML Schemas are compatible and could be integrated with XML Schema integration methods. Each element is described in detail in the paragraph 7.3. The major design aspects are illustrated in this paragraph for a better understanding of the relationships.

The Terms-Of-Use information model and the additional elements could be considered as GeoDRM capabilities. This point of view has the advantage that the established OGC basic service model/common operation “getcapabilities” and the capabilities document could be used. Because this operation has no legally binding impact, it also satisfies the “fully informed” requirements (3.7). The different embedding strategies of the root element <GeoDRMCapabilities> are proposed in 7.4.

The XML schema was also used from a processing point of view with the operations

- Getcapabilities
- NegotiateTerms

with request and response pairs. The operation negotiate terms was overloaded. It seems advantageous to split this overloading into a third operation:

- acceptPrecondition

The operations are using subsets and are described in the XML Schema.

7.2.1 Main Axis

The GeoDRMCapabilities element encapsulates all required information for a successful business function request. The information model re-uses the OWS Common Capabilities Type and adds the products catalog. The CapabilitiesType is re-used for the stand-alone variant to describe the business operations and to identify the service. The product catalog encapsulates the hierarchic product elements. Figure 19 depicts the relationships.

⁶ OGC Document 05-008r1

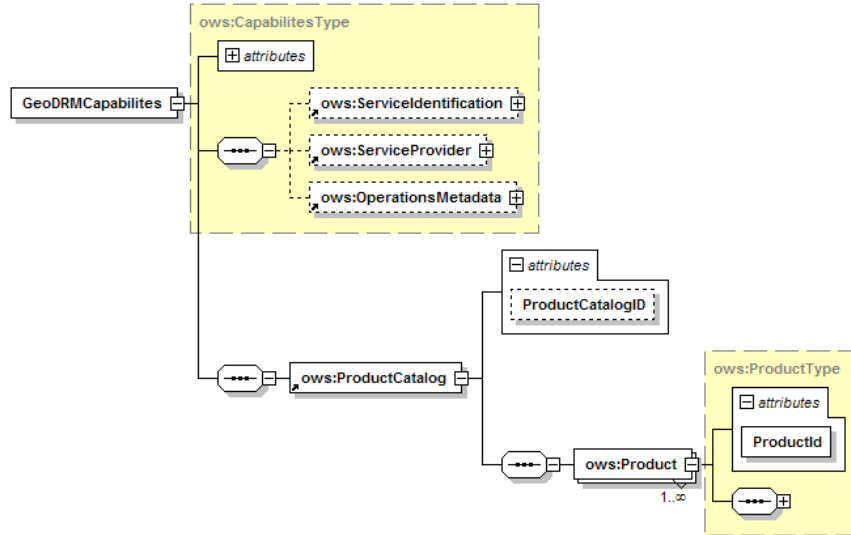


Figure 19: Main Axis

Note: A powerful inheritance mechanism may be useful between product instances and is subject for future work. The advantages are shown in the OGC WMS specification with the <layer> element.

7.2.2 Matching: Product – Resources

Paragraph 7.1 requires a clear separation between the more general product elements and its concrete resources. A product may encapsulate a number of WMS layers or a combination between WMS layers and WFS features. It is also possible to have two separated products but the same resource. This is the case if different terms-of-use apply, e.g. a product with support and another without support. Figure 20 depicts a product resource. A resource may have multiple resource records. Each record may have resource capabilities. This node may embed a capabilities document of a content service. The embedded capabilities are necessary in the case that the regular capabilities of a content service are not accessible from the internet. This happens if the content service is behind a firewall and only accessible via the business services to enforce the correct use of business functions.

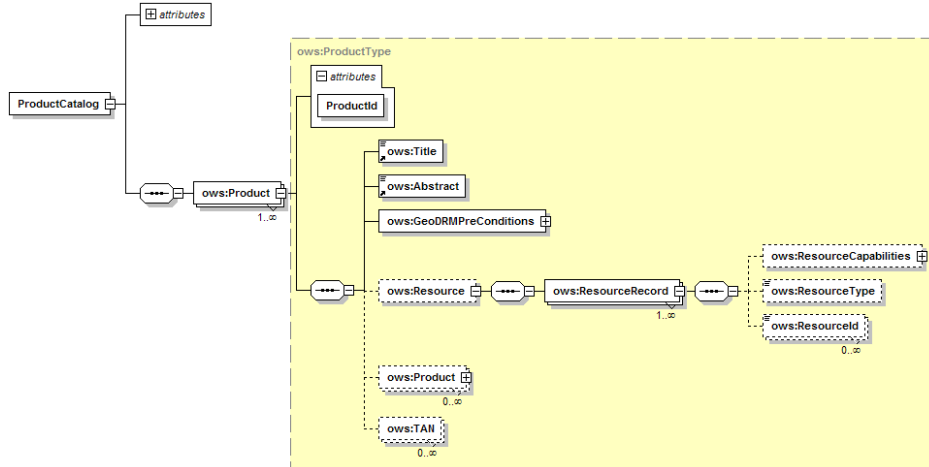


Figure 20: Product resources

The Resource Type is a general term to express the object name. In an OGC WMS example, a resource type is “layer”, the identifier ResourceID may be the layer “water”. Table 5 gives some examples.

	ResourceType	ResourceID
WMS 1.0.0	layer	“Water”
WFS 1.0.0 with GML 2	featureid	“Water”
WFS 1.0.0 with GML 3.0.1.	fid	“water”
A complete service with all content W*S with BSM description	name	myWMS
A complete service with all content W*S with OWS Common	Ows:title	myWMS

Table 5: Different matching examples⁷

Wildcards can be used to express grouping in a simplified form, which lowers the maintenance efforts. Table 6 shows examples for wildcards. In the WMS case all layers are related to the product and its preconditions. New content may be added as a layer without a need to adjust the business functions. The second example shows the advantage that content may be covered by business functions without service specific distinctions. In this case the product “water” relates to all data sets with the name “water”. The last example is a service farm case. All content services may be served with a regular disclaimer.

⁷ Remark: Table 1 shows the large range due to historical and other reasons. Maybe this view will help to harmonize and precise the description of a service. Identifiers are helpful to manage service farms.

	ResourceType	ResourceID
WMS 1.0.0	layer	*
WMS 1.0.0	*	“Water”
WFS 1.0.0 with GML 2	*	“Water”
WFS 1.0.0 with GML 3.0.1.	*	“Water”
A complete service with all content W*S with BSM description	name	*

Table 6: Different matching with wildcards

7.2.3 Preconditions

The given uses cases, e.g. authentication & ToU or authentication & pricing, show that different business models may use only a subset of business functions. Therefore each offer needs a defined set of preconditions. Because of the open list of business functions, the modeled list should only be considered as a placeholder. Currently “Authentication” and “TermsManagement” are subject for specification. It might be possible to offer multiple types of preconditions functions. An example is to offer a SOAP authentication and a HTTP GET authentication method in parallel. A business client can choose a preferred mechanism, e.g. depending on its binding capacities or on a security level. Figure 21 illustrates the relationships.

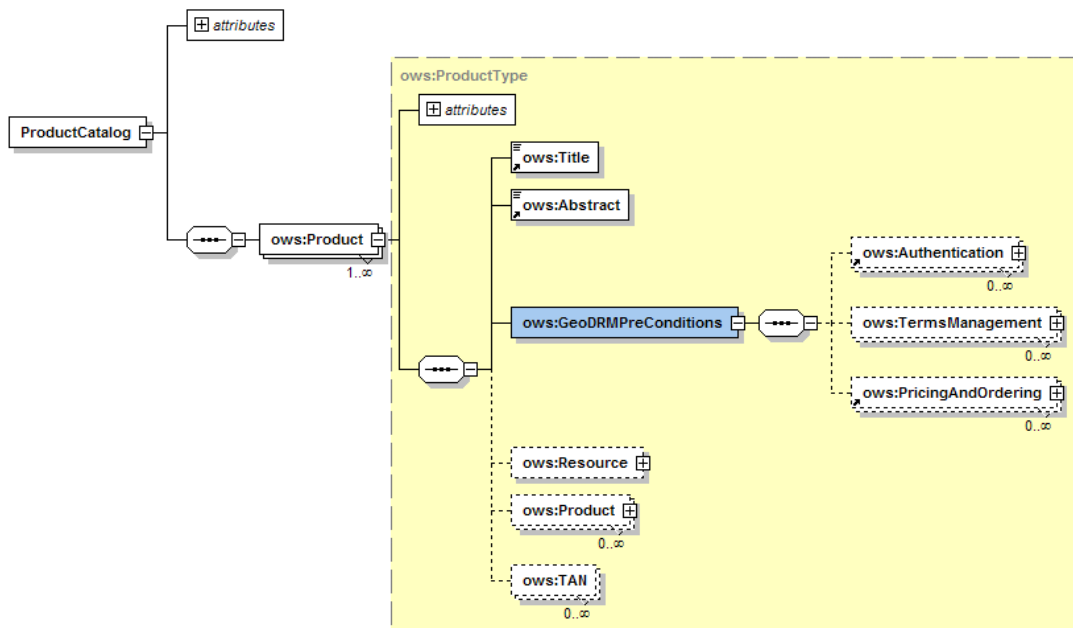


Figure 21: PreConditions

7.2.4 Terms Management

This overview focuses on the terms management function. It consists of two parts, which are displayed in Figure 22. The first element “terms” covers the Terms-of-Use information resource. It is possible to embed it or to reference it. The second part is the required workflow for a successful access. The workflow element serves for many business functions and is explained in detail in the next paragraph.

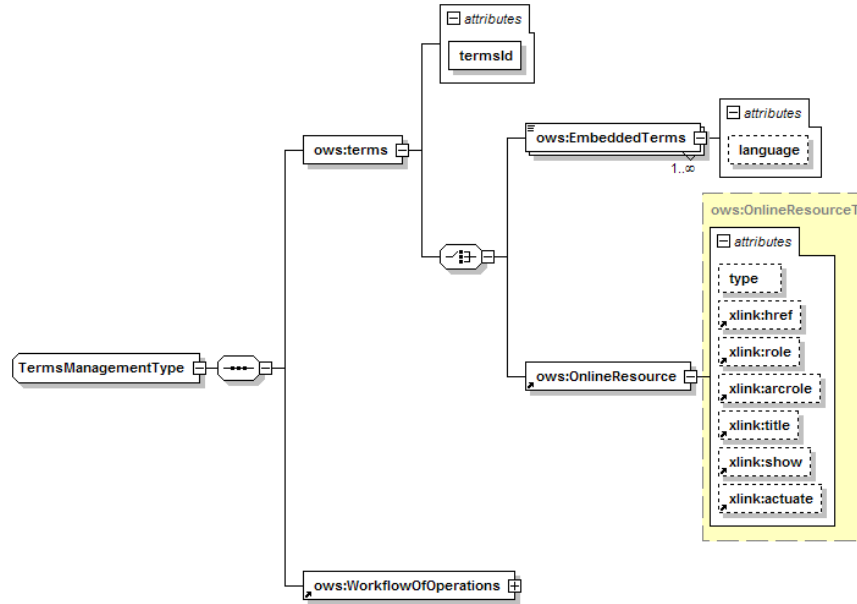


Figure 22: Terms Management

7.2.5 Workflow

The former OGC Basic Service Model and the current OGC Common Service Model have extensive methods to describe operations and its interfaces. These abilities should be re-used. Therefore the detailed operation description will be stored in the regular capabilities document. The detailed operation description can be re-used multiple times by referencing it. This could be done for a product and its preconditions with the “WorkflowOfOperationsType” element, depict in Figure 23. It contains an optional description of the protocol. It might be necessary to have an ordered workflow with more than a single operation to fulfill the task.

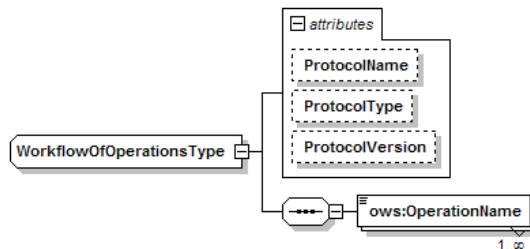
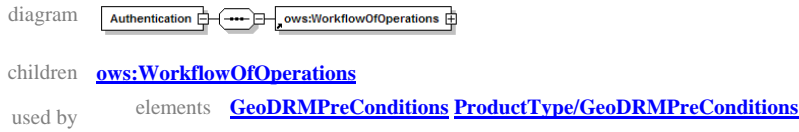


Figure 23: Workflow element

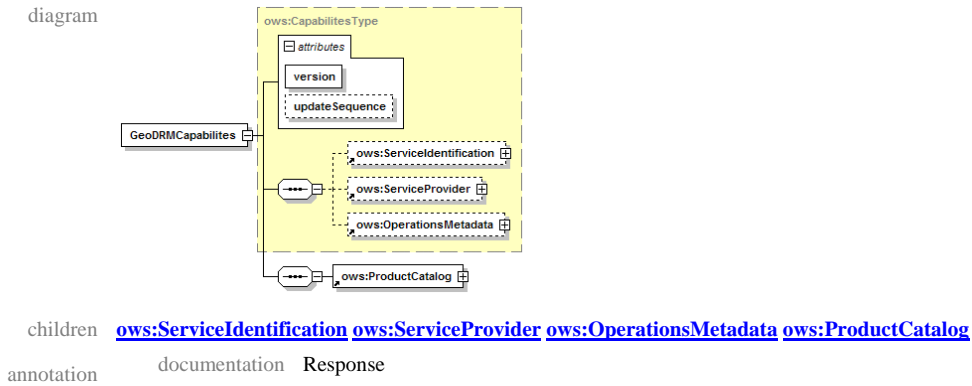
7.3 Schema Design

This chapter describes the used elements for GeoDRM in detail. Some OWS Common elements are described here, but most are neglected because of repetition. Many elements have self describing names.

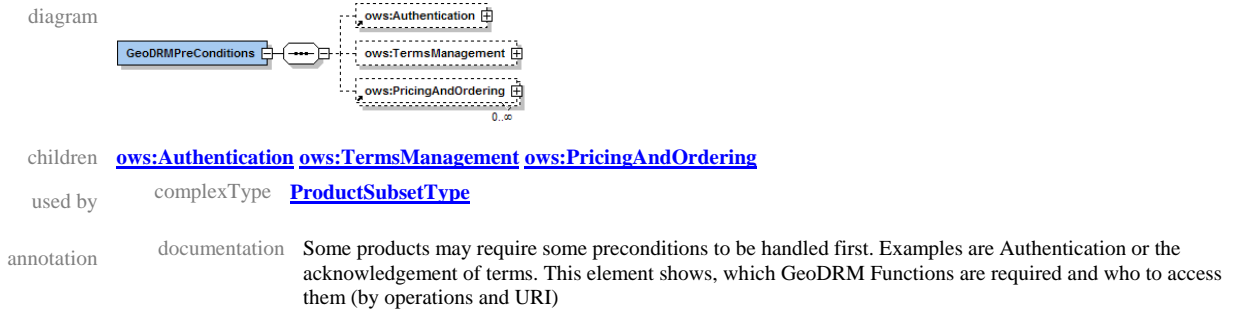
7.3.1 Element Authentication



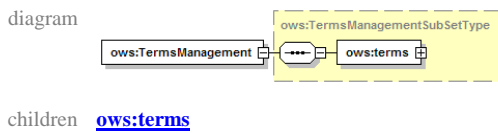
7.3.2 Element GeoDRMCapabilities



7.3.3 Element GeoDRMPreConditions

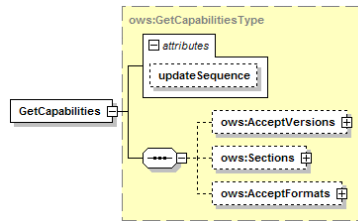


7.3.4 Element GeoDRMPreConditions/TermsManagement



7.3.5 Element GetCapabilities

diagram

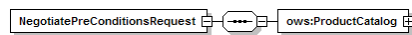


children [ows:AcceptVersions](#) [ows:Sections](#) [ows:AcceptFormats](#)

annotation documentation Request

7.3.6 Element NegotiatePreConditionsRequest

diagram

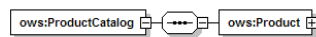


children [ows:ProductCatalog](#)

annotation documentation Request

7.3.7 Element NegotiatePreConditionsRequest/ProductCatalog

diagram

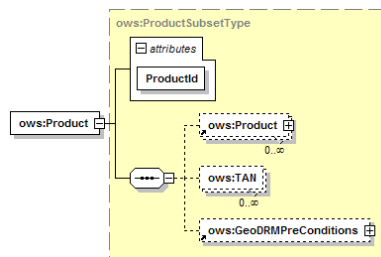


children [ows:Product](#)

used by elements [GeoDRMCapabilities](#) [NegotiatePreConditionsResponse](#)

7.3.8 Element NegotiatePreConditionsRequest/ProductCatalog/Product

diagram

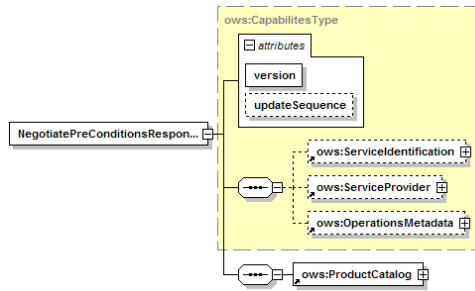


children [ows:Product](#) [ows:TAN](#) [ows:GeoDRMPreConditions](#)

used by complexType [ProductSubsetType](#)

7.3.9 Element NegotiatePreConditionsResponse

diagram

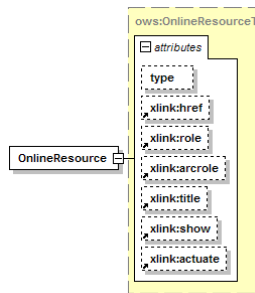


children [ows:ServiceIdentification](#) [ows:ServiceProvider](#) [ows:OperationsMetadata](#) [ows:ProductCatalog](#)

annotation documentation Response

7.3.10 Element OnlineResource

diagram

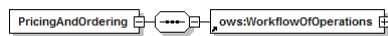


used by elements [ProductType/Resource/ResourceRecord/ResourceCapabilities](#) [TermsManagementType/terms](#)

annotation documentation On-line information that can be used to contact the individual or organization. OWS specifics: The xlink:href attribute in the xlink:simpleLink attribute group shall be used to reference this resource. Whenever practical, the xlink:href attribute with type anyURI should be a URL from which more contact information can be electronically retrieved. The xlink:title attribute with type "string" can be used to name this set of information. The other attributes in the xlink:simpleLink attribute group should not be used.

7.3.11 Element PricingAndOrdering

diagram

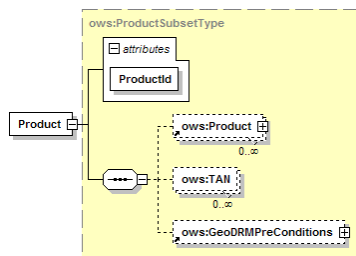


children [ows:WorkflowOfOperations](#)

used by elements [GeoDRMPreConditions](#) [ProductType/GeoDRMPreConditions](#)

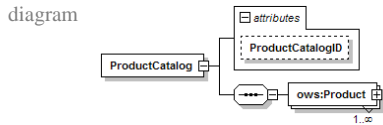
7.3.12 Element Product

diagram



children [ows:Product](#) [ows:TAN](#) [ows:GeoDRMPreConditions](#)
 used by complexType [ProductSubsetType](#)

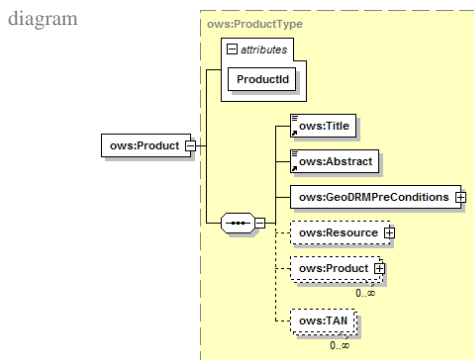
7.3.13 Element ProductCatalog



children [ows:Product](#)
 used by elements [GeoDRMCapabilities](#) [NegotiatePreConditionsResponse](#)

annotation documentation The Tag ProductCatalog is the root element for the product tree. This element helps to structure multiple catalogs from different vendors

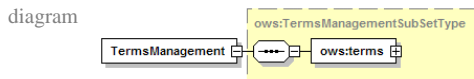
7.3.14 Element ProductCatalog/Product



children [ows:Title](#) [ows:Abstract](#) [ows:GeoDRMPreConditions](#) [ows:Resource](#) [ows:Product](#) [ows:TAN](#)
 used by complexType [ProductSubsetType](#)

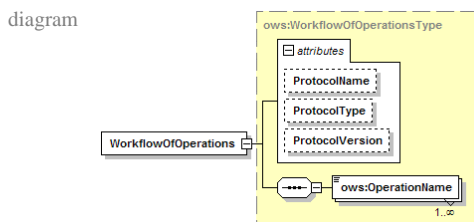
annotation documentation The business expression "product" helps to abstract concrete layers, features, fids, services. A product is a defined offer from a business point of view and has (business) terms.

7.3.15 Element TermsManagement



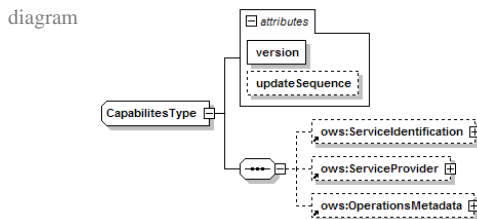
children [ows:terms](#)

7.3.16 Element WorkflowOfOperations



children [ows:OperationName](#)
 used by elements [Authentication](#) [PricingAndOrdering](#)
 complexType [TermsManagementType](#)

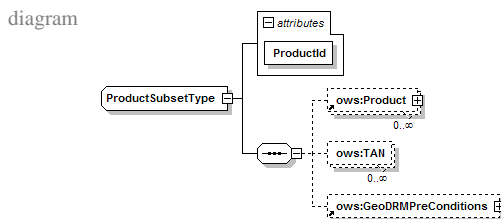
7.3.17 ComplexType CapabilitesType



children [ows:ServiceIdentification](#) [ows:ServiceProvider](#) [ows:OperationsMetadata](#)
 used by elements [GeoDRMCapabilites](#) [NegotiatePreConditionsResponse](#)

annotation documentation XML encoded GetCapabilities operation response. This document provides clients with service metadata about a specific service instance, usually including metadata about the tightly-coupled data served. If the server does not implement the updateSequence parameter, the server shall always return the complete Capabilities document, without the updateSequence parameter. When the server implements the updateSequence parameter and the GetCapabilities operation request included the updateSequence parameter with the current value, the server shall return this element with only the "version" and "updateSequence" attributes. Otherwise, all optional elements shall be included or not depending on the actual value of the Contents parameter in the GetCapabilities operation request.

7.3.18 ComplexType ProductSubsetType



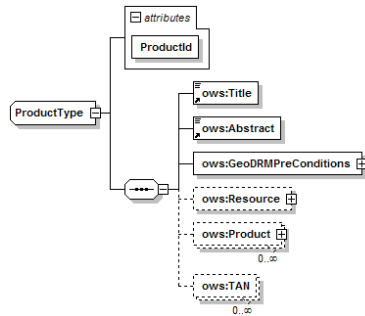
children [ows:Product](#) [ows:TAN](#) [ows:GeoDRMPreConditions](#)
 used by elements [NegotiatePreConditionsRequest/ProductCatalog/Product](#) [Product](#)

7.3.19 Element ProductSubsetType/TAN



7.3.20 ComplexType ProductType

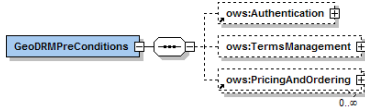
diagram



children [ows:Title](#) [ows:Abstract](#) [ows:GeoDRMPreConditions](#) [ows:Resource](#) [ows:Product](#) [ows:TAN](#)
 used by elements [ProductCatalog/Product](#) [ProductType/Product](#)

7.3.21 Element ProductType/GeoDRMPreConditions

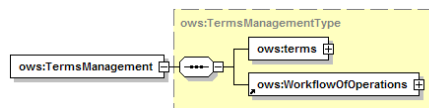
diagram



children [ows:Authentication](#) [ows:TermsManagement](#) [ows:PricingAndOrdering](#)
 used by complexType [ProductSubsetType](#)

7.3.22 Element ProductType/GeoDRMPreConditions/TermsManagement

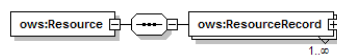
diagram



children [ows:terms](#) [ows:WorkflowOfOperations](#)

7.3.23 Element ProductType/Resource

diagram

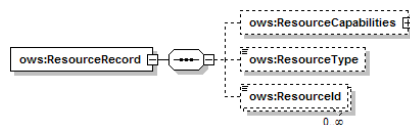


children [ows:ResourceRecord](#)

annotation documentation A product is a business definition. It is derived from resources, e.g. services or data.

7.3.24 Element ProductType/Resource/ResourceRecord

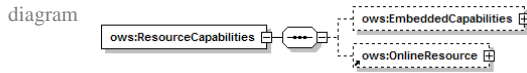
diagram



children [ows:ResourceCapabilities](#) [ows:ResourceType](#) [ows:Resourcecld](#)

annotation documentation A product definition may consist of multiple ResourceRecords, e.g. multiple services. In a simple case it is just a WMS layer.

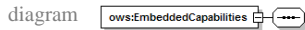
7.3.25 Element ProductType/Resource/ResourceRecord/ResourceCapabilities



children [ows:EmbeddedCapabilities](#) [ows:OnlineResource](#)

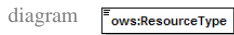
annotation documentation A product resource definition (OGC OWS common Capabilities) might be embedded in the product definition or referenced. In the case that a product is referenced within the XML document, the ResourceCapabilities Element is optional.

7.3.26 Element ProductType/Resource/...../ResourceCapabilities/EmbeddedCapabilities



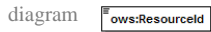
annotation documentation In the case of embedded OGC OWS common capabilities, the complete XML document is stored in the GeoDRM Capabilities Document. In the case of a reference only the URL need to be defined in the OnlineResourceblock

7.3.27 Element ProductType/Resource/ResourceRecord/ResourceType



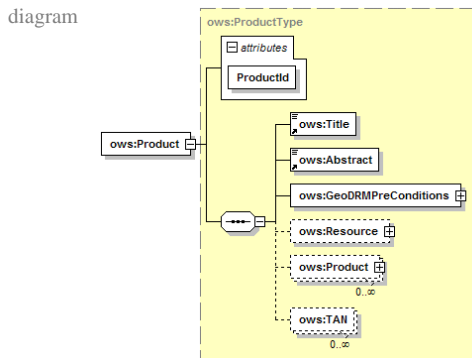
annotation documentation XML Tag name, e.g. layer, fid

7.3.28 Element ProductType/Resource/ResourceRecord/ResourceId



annotation documentation XML Tagfield Value, e.g. water, "123"


7.3.29 Element ProductType/Product



children [ows:Title](#) [ows:Abstract](#) [ows:GeoDRMPreConditions](#) [ows:Resource](#) [ows:Product](#) [ows:TAN](#)


used by complexType [ProductSubsetType](#)

7.3.30 Element ProductType/TAN

diagram 

 annotation documentation Some Business Models have higher trust requirements. Therefore TransactionNumbers (TAN) can be used for a higher level. The mechanism is often used for home banking. A user might acquire the TANs by various ways. A defined ways are: 1.) Publishing in the Capabilities document as a static or dynamic value, 2.) delivery by GeoDRM response

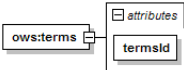
7.3.31 ComplexType TermsManagementSubSetType

diagram 

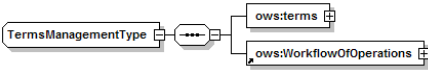
 children [ows:terms](#)

 used by elements [GeoDRMPreConditions/TermsManagement](#) [TermsManagement](#)

7.3.32 Element TermsManagementSubSetType/terms

diagram 

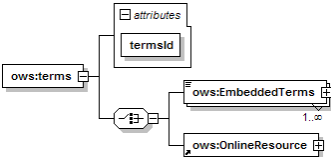
7.3.33 ComplexType TermsManagementType

diagram 

 children [ows:terms](#) [ows:WorkflowOfOperations](#)

 used by element [ProductType/GeoDRMPreConditions/TermsManagement](#)

7.3.34 Element TermsManagementType/terms

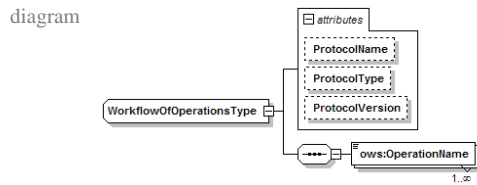
diagram 

children [ows:EmbeddedTerms](#) [ows:OnlineResource](#)

7.3.35 Element TermsManagementType/terms/EmbeddedTerms

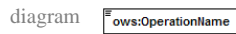
diagram 

7.3.36 ComplexType WorkflowOfOperationsType



children [ows:OperationName](#)
used by element [WorkflowOfOperations](#)

7.3.37 Element WorkflowOfOperationsType/OperationName



7.4 Embedding Methods

The requirements of the OWS3.geoDRM initiative resulted in new functions and information models. These parts need to be embedded into the existing OGC environment.

7.4.1 Backwards compatible embedding

After the release of the popular OGC WMS specification in 2001 many OGC and non-OGC members developed compatible products. Many data providers are using these products for very individual purposes. Therefore many services are already up and running.

The new business functions might be also relevant for these content service instances. Because the majority of old clients do not perform XML Schema validation, a suitable way to integrate the information model is by extending the BSM defined “capabilities” document. Figure 19: Main Axis shows the added tag “productcatalog”, which includes all necessary terms-of-use information. Old clients will ignore this tag and can continue interaction with the service. Terms-of-use enabled content clients or Terms-of-Use client can use the information and perform suitable operations requests.

7.4.2 Current and upcoming specification embedding

Future specifications should consider the XML node “productgroup” by referencing it in the XML schema definition. OGC implementation specification can embed them by the XML schema “include” mechanism. If geo-eBusiness functions are considered in general as widely required, OWS common should include these XML schemas.

8 Common Elements

Although the described implementations and the Terms-of-Use model in chapter 4-7 offer different possible implementation solutions, many concepts and elements are in common. These are described in this chapter on a more abstract level than the HTTP GET or SOAP level. The common concepts and elements can be distinguished into process and information models.

8.1 Process Model

In general the classic trading process with a

- information phase,
- negotiation phase,
- contracting phase and
- delivery phase

can also be found in the implementations approaches. The phases are described in detail below. Figure 24 illustrates the phases and introduces some sub processes. Of course the number of sub processes is open. The path 1 shows the trail & error path.

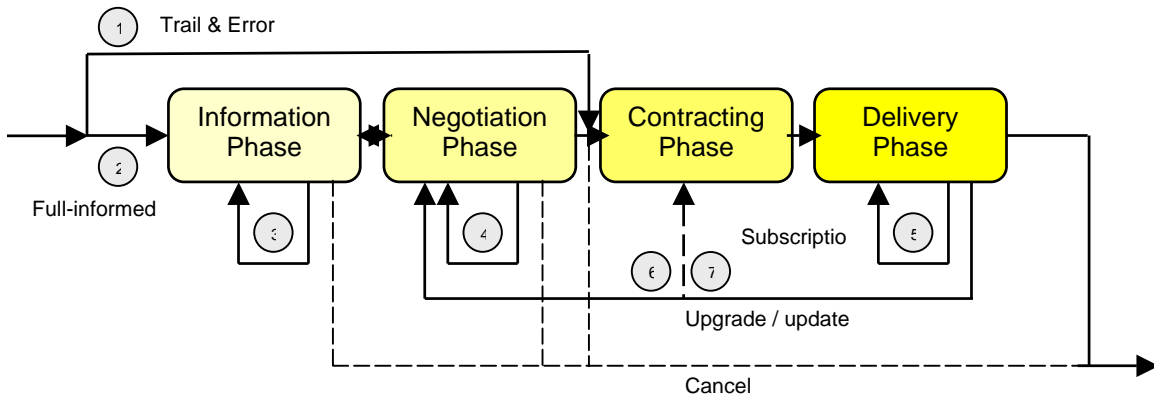


Figure 24: Workflow element

The information and negotiation phase are intentionally skipped and a potential rejection is prepared. Path 2 shows the full-informed process. A user might check out multiple product descriptions without any further actions (Path 3). If there is an interest the next step is the negotiation phase. Additional preconditions may depend on user's selection or input. The negotiation can be repeated with other selected values (path 4). After the legally binding contracting phase the product will be delivered. If a user ordered a subscription, there might be multiple delivery instances, e.g. flat rate for a year or five times (path 5). Many providers give better conditions for additional business transactions, e.g. upgrade or update. Path 6 and 7 show the starting point. Path 6 shows an additional negotiation, e.g. to receive a special price. Path 7 shows a direct contracting in the case, if a user has a framework contract [without a flat rate], so each request need to be ordered and e.g. different prices per request depending on the request parameters may apply. An example for the last path is a mobile telephone (framework) contract [not a flat rate offer]

with a detailed billing list. In this case each call may result with a different price, but there is no additional negotiation phase.

Of course a user can abort the overall process prior the contracting. If he did not find interesting offers, he may already leave after the information phase. Also the path 1 (trail & error) may result with a rejection.

Some implementation approaches in OWS3 skipped a phase or combine two phases in a single request & response pair. Because the basic underlying protocol http is stateless, each received request need to be examined. Or vice versa: each phase can be addressed (and rejected) directly. Attached credentials can be used to store and pass already reached states.

8.1.1 Information phases

Chapter 7 describes enhanced elements for the capabilities in detail, which includes published Terms-of-Use and the Access Control mechanisms. The implementation approach described in chapter 6 also remarks the need to express that a service point is “DRM-enabled”, which type of authentication method is accepted and information about additional operations. This approach introduces an operation *GetLicense*, which returns detailed information about access constrains and also includes the Terms-of-Use text. The implementation approach described in chapter 4.6.2 uses the known OGC BSM tag `<AccessConstraints>` to indicate that additional access constrains (authentication and Terms-of-Use) are required. The implementation approach described in chapter 5 requests also that the capabilities document needs to describe that preconditions exist (See 5.5).

8.1.2 Negotiation phase

In some cases additional contract elements are required than the published elements. Examples might be existing and/or user specific licensing contracts (numbers)(see figure 24, path 6). Usually a negotiation phase is considered as a phase without any legally binding results. Because of the general assumption, any GeoDRM business solutions should respect this common understanding.

The implementation approaches in chapters 4-6 created very different solutions. In general the solutions do not consider clearly the different intentions between negotiation and agreement. From a technical point of view both solutions might be equal, but from a legal point of view there is a significant difference. The approach described in chapter 5 does not provide any engine executable solution for the negotiation phase. It assumes that an out-of-band mechanism (see 5.2) is handling this phase, e.g. on a human readable website. The approach of chapter 6 introduces an operation `NegotiateTerms`. This operation includes earlier proposed operations known as “`doLicenseAgreement`” or “`agreeToLicense`”. The last proposal mixes the negotiation and contracting phases. On the other hand the workflow supports the negotiation phase also in an engine executable way. It passes the expert service request (e.g. `getmap`) with the parameter `OWSREQUEST=owsRequestName` already in the negotiation phase to the GeoDRM point, which allows detailed examination.

The implementation approach described in chapter 4 uses the operation “`GetUnsatisfiedPreconditions`” and a flag mechanism “`-check_only`” (Testing Access 4.5.1) to express that the request should not be considered for (legally binding) execution. The response returns the simulated result, e.g. access would be granted or not.

8.1.3 Contracting phase

If all required conditions are fulfilled and the user actively “signs” the (negotiated) contract by pushing a specific button, the access will be granted. As a result of the binding status a token is returned as a handle for further negotiations and transactions under the same conditions. From a legal point of view, the result of this contracting phase is a license. The implementation approach of chapter 4 uses a semi out-of-band process with browser re-direction commands and cookies to store the accepted contracts. The service site returns each time also the already accepted contracts within the `<Accepted>` element.

The implementation approach of chapter 5 considers also the contracting phase as an out-of-band mechanism analogue to its negotiation phase solution. The results of the out-of-band process are credentials for the upcoming delivery phase (see 5.2, 3).

The implementation approach of chapter 6 separated initially the negotiation phase and the contracting phase. The operation “`agreeToLicense`” represents the contracting phase. Therefore the user executes explicitly this operation with suitable contract references (`LICENSEIDS`). The operation `NegotiateTerms` is surcharged to handle also the contracting phase in the final solution of the development.

8.1.4 Delivery phase

The last phase of the transaction process is the delivery phase. Although a contract can only be agreed once, the delivery phase may be executed multiple times depending on the business model. Some negotiated and contracted conditions may expire. Therefore all delivery requests should be examined. In an example business cases, the user may have the right to request fresh data by instant delivery for a time frame of a year or without limits.

On the other hand it may be important for a correct accounting to track the delivery and start other processes (e.g. billing) only if the product is delivered successfully.

All OWS3.geoDRM implementations support online delivery. The delivery process will be started if the required credentials are attached to the delivery request and are valid. The delivery phase request is not carried out separately, but mentioned explicitly in the description (Example: 5.2 /5,).

8.2 Information Model

Different information models are used to describe the elements Terms-of-Use, user identification and the resulting license. The approaches described in chapter 4 and 6 show that the same information structures (XML Schema) can be used for multiple requests, only with a different degree of completeness. The used information models can be structured into the following elements.

8.2.1 Capabilities

The capabilities information model is used for the information phase. It contains service binding and content information. In the case of Terms-of-Use click-through service it needs to indicate that a service point with additional GeoDRM workflows information, e.g. authentication and terms-of-use acceptance. The implementation solutions neglected in general the detailed descriptions of the service point capabilities in the getcapabilities document. But it was stated that a detailed description is required. A potential solution approach is provided in chapter 7 in detail.

The implementation approach of chapter 4 re-uses the OGC BSM known tag `<AccessConstraints>` (see 4.6.2) to indicated current and older clients the need for additional workflows prior access. The implementation of chapter 5 uses the BSM or OGC OWS common capabilities to describe the `getSession` operation (5.5.3), but adds new tag elements (`<TokenType>`) and is therefore also not complete compatible to BSM or OGC OWS common. Chapter 6 states that three key elements need to be described in the capabilities (DRM-enabled, accepted authentication methods, description of operation binding).

8.2.2 User Identification Model

The given use cases require information models for anonymous or identified users. Another scenario is the state less case for file download, which requires no user information model at all.

All three implementation separated the security information clearly to other (e.g. terms-of-use) information. As mentioned, chapter 6 also separated consequently the operations (`getSession`, `doService`).

Although the separation is in common the underlying security models have different encodings. The implementation of chapter 4 uses the DACS definition (`DACS:federation-name:jurisdiction-name:username=value`). Chapter 5 has a generic approach and supports four token profiles (Username Token Profile, X.509 Token Profile, SAML Token Profile, REL Token Profile). Chapter 6 references the GDI NRW Web Security Service approach which uses SAML encodings.

Therefore more than five encodings for security are referenced within OWS3.

8.2.3 Terms-of-Use Model

In opposite to the user identification model, no implementation referenced the terms-of-use model to other standards institutions. The implementation of chapter 4 and 6 need only a unique Terms-of-Use Model ID and a text or URL (see 4.6.4, 6.1.4). The implementation of chapter 5 does not handle terms-of-use information models at all, because the management is out-of-band.

8.2.4 License Model

As outlined above, the result of the negotiation and contract phase with the terms-of-use is a license. Terms-of-use may be considered as a license proposal. Although the difference is not handled very carefully and the naming is even completely wrong, the implementations show the difference. Chapter 4's approach will be used as an example:

The implementation approach of chapter 4 releases after a successful negotiation and contracting phase a tag `<License id="uniqueId99">` and a "Notice Acknowledgment Token" (NAT, see 4.7.1). The resulting NAT can be used to start the delivery phase. The NAT may also be used for another (information,) negotiation and contracting phase workflow (see 8.1.2 additional contract elements). Many business models offer better conditions to customers with previous contracts (e.g. for update, upgrade, framework contract...). The approach of chapter 4 offers also a "Notice Acknowledgment Token" integration mechanism (4.7.4.2.2), which may be used to integrated contracts.

8.3 Rejection Mechanism

All implementations have a rejection mechanism, if the attached credentials are missing or are not valid. In two implementations the rejection message has more information than a regular OGC exception, it contains an URL or a re-direct command to a point, which handles the negotiation phase (again). This possibility is necessary for the trial & error approach. It attempts a successful walk through, but is prepared for rejection (See 4.2; 5.2, 6.2.5).

8.4 Session Establishment Mechanism

Because the HTTP protocol is stateless and the OWS3.geoDRM goal Terms-of-Use click-through service requires more than a single operation, a session mechanism is required. The indirect session mechanism via authentication and password is not sufficient, because anonymous users should be supported, too (see use case 2.1 and requirements 3.2).

The implementation solutions described in chapter 4 uses the HTTP cookie mechanism to establish a session management. This approach works well in WWW browser environments, if the user configured its browser to accept cookies. Most browsers have menus to switch off cookie usage.

The implementation solution described in chapter 5 introduces in paragraph 5.5.1 an explicit request `getSession` with different types of tokens (password or session). This operation is described in the capabilities 5.5.3.

The approach in chapter 6 re-uses the session mechanism specified [2] in the GDI NRW Web Security Service (WSS) protocol, which is used as an underlying protocol. The GDI NRW WSS protocol offers an explicit and optional `getSession` operation.

9 Conclusions

OWS3.geoDRM was an initial step into business, legal and software architectural related issues. Although the task seemed to be limited, it raised many open questions in the mentioned fields.

A key question is the relationship to third-organization standards, e.g. for security issues. There are multiple solutions for different purposes available. In many environments, some specific solutions are mandatory. If the diversity of e.g. security solutions can not be harmonized, a **negotiation framework mechanism** for **multiple bindings** might be an interoperable bridge.

OWS3.geoDRM showed that **multiple kinds of functions** need to be integrated (Term-of-Use, authentication, content services). More functions (e.g. pricing & ordering, encryption) are already in sight. Although the functions are different, the required **embedding** and software packaging mechanisms are similar or even the same. A general framework mechanism may encapsulate the embedding mechanism and information model transformation (“facades”) to avoid repetition.

Chapter 8 introduced the classic trading phases:

- Information Phase
- Negotiation Phase
- Contracting Phase
- Delivery Phase

These phases apply also to the OWS3.geoDRM cases. Although the implementation solutions used may different operation names, it seems that a neutral operation name for each phase is helpful.

The neutral term should also be applicable for the following business functions:

- Authentication
- Terms-of-Use
- Web Pricing & Ordering
- *...other upcoming functions*

9.1 Relationship between general business properties and business functions

OWS3.geoDRM showed that different business functions need to be integrated in different phases. The concrete selection depends on the concrete business model and is therefore provider specific. Because all functions need to fit into the general business phase model, it is proposed later in this chapter to define a general set of operations and information elements for the general process between the phases. It is expected that the descriptions of the business functions could be express with a general pattern. Same applies to the required workflow to chain the functions. Another example for a general property for all business function is the productID.

Figure 25 shows the relationship between a general model and the derived functions. This approach is suitable to express also relationship/dependencies between the functions in a harmonized way. It also reduces repetition, e.g. embedding, as described above in detail.

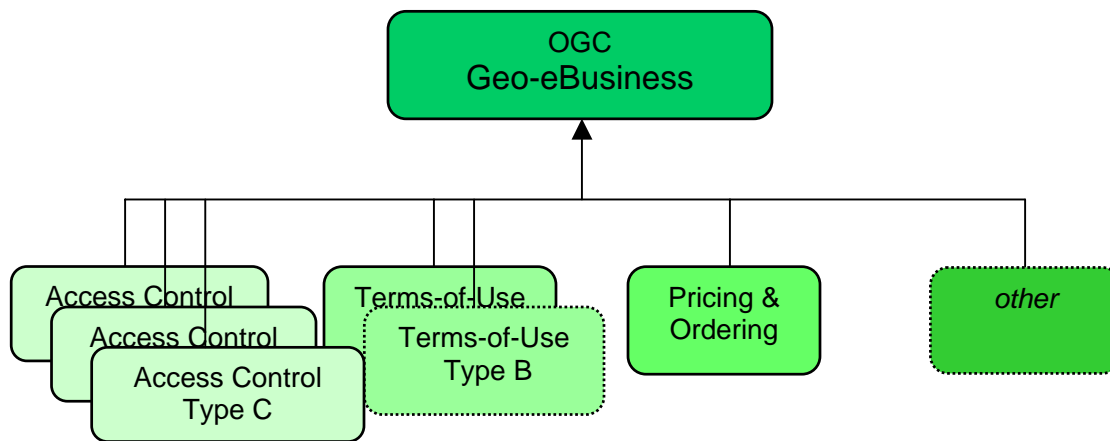


Figure 25: Inherited general business function elements

If SDI operators and vendors can not agree on a specific type of business function, e.g. Access Control, it may be also possible to permit multiple types of the same business function. A data provider may operate also multiple types in parallel. Figure 25 illustrates that with Access Control and Terms-of-Use.

9.2 Proposed General Phases and Tracks

Paragraph 8.1 described the classic business phase model starting with the information phase and finalizing with the delivery. Although the concrete use is depending on a concrete business model and derived business processes, the business functions (Access Control, Terms-of-Use...) can be considered as separated tracks crossing the phases. Figure 26 illustrates the functions and the content “traveling” through the phases. As mentioned above, it is not possible that a function will not be used in a specific phase or even not at all.

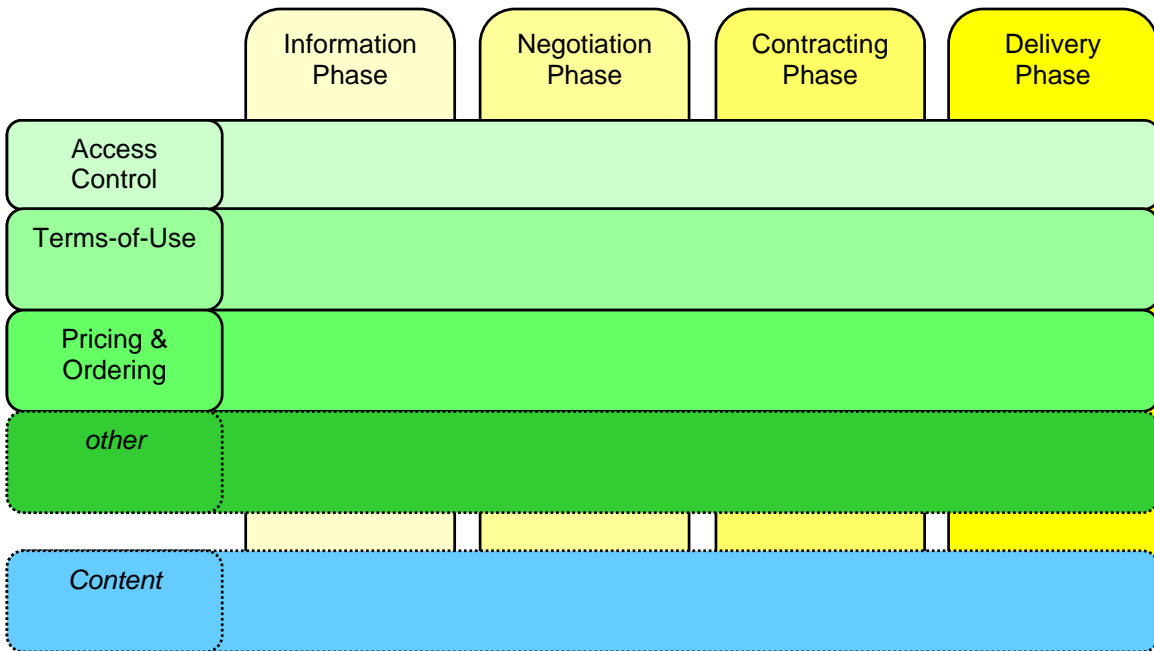


Figure 26: General business functions and information phases

OWS3 focused on the Terms-of-Use Click Through application. This business function Terms-of-Use was supported with Access Control for better convenience. The content was delivered after the user agreed to the Terms-of-Use. Figure 27 shows a concrete workflow routing for the OWS3 use cases 1-3 (see chapter 2).

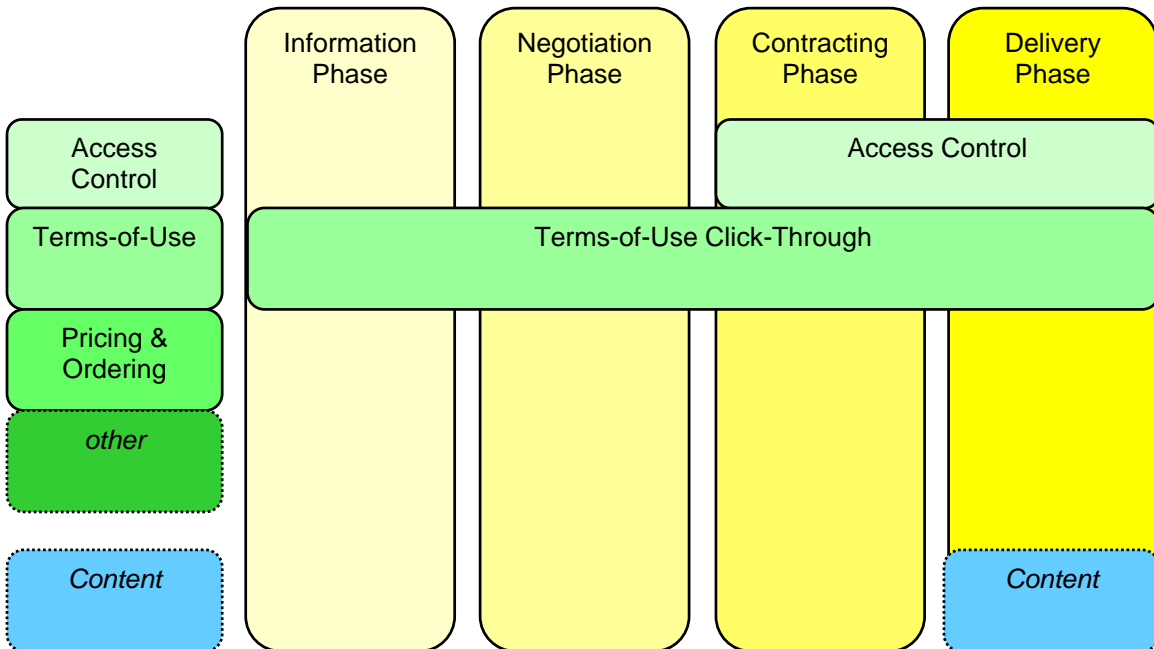


Figure 27: OWS3 Use Cases 1-3

Another example might be a typical “eShop” with pricing & ordering. In this case a user can receive some prices already in the information phase. Many pricing models give the user a choice, which may result in different prices. Offers are given in the negotiation phase, but without any legal binding results. Terms-of-use apply also in three phases. If the price is acceptable, a user can order the product in the contracting phase. But in this phase, Access Control is required to identify the user or to register him. After the content is delivered, the pricing & ordering function can create a billing record.

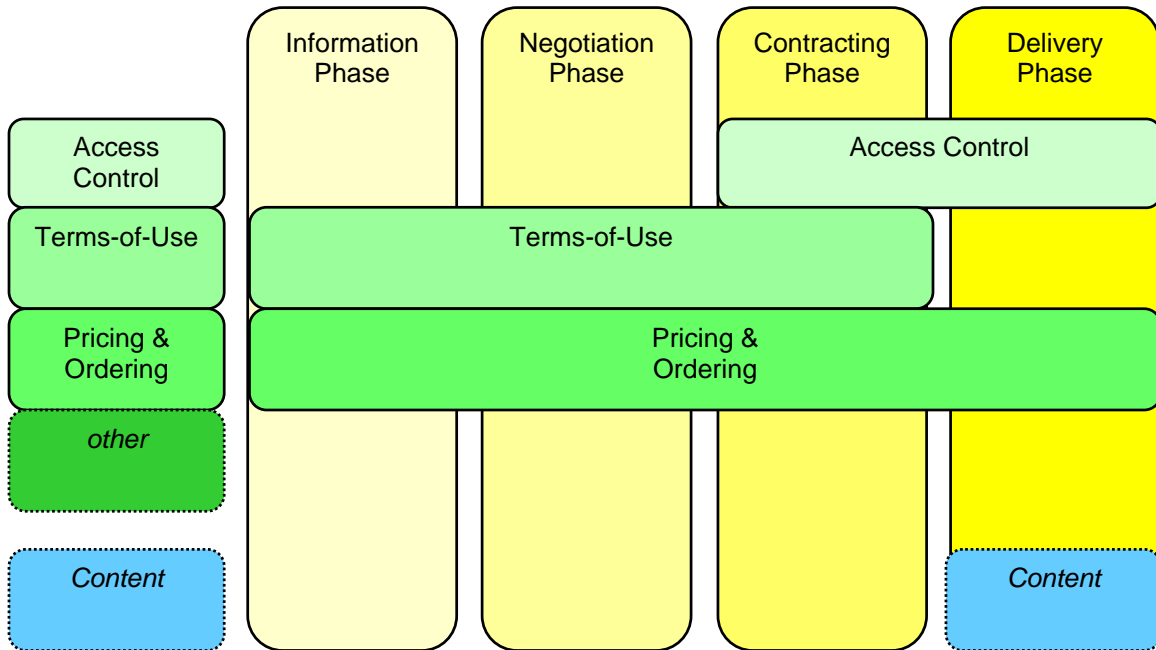


Figure 28: Business function in an eShop Scenario

9.3 Proposed operations

The following operations are proposed as a conclusion. Although the shown examples above are assumptions and were not discussed in detail, a conclusion of OWS3 is that a set of business function has to be processed in a single step. An example is use case 3a (see 2.3, named user) and it requires user identity check with Access Control and also the Terms-of-Use check. Therefore, the service gateway needs to receive user identity information (e.g. login/password) and Terms-of-Use instance information (TermsOfUseID) together. The analyses of both information fragments is not subject for standardization and can be processed in a “black box” approach. The response needs to be packed in an interoperable encoding.

It might be possible that more specific operations are required. The proposed operations cover the classic phase workflow. Figure 29 illustrates the operations in the context of business operations and phases.

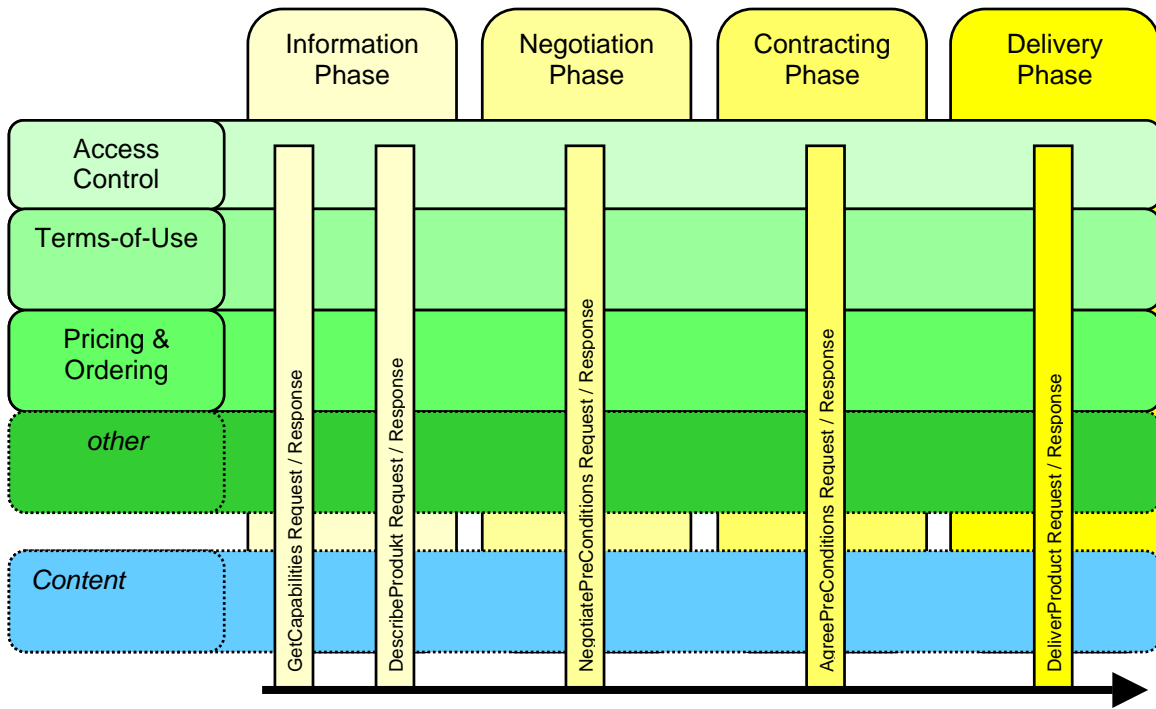


Figure 29: Proposed operations corresponding to the information model containing sub data models for business functions

The operations are described in detail in the following paragraphs.

9.3.1 Operation: getCapabilities ()

The basic operation returns information about the operation bindings and a list of products with product IDs. A hierarchical grouping structure is helpful.

Because multiple kinds (e.g. Access Control, terms of use, pricing) and types (e.g. Access Control type A for HTTP GET, Access Control type B for HTTP SOAP, ...) of functions need to be chained, the workflow and the process order need to be defined and described. It should be possible to offer multiple types in parallel.

Table 7 shows potential elements.

GetCapabilities	general	Request	Response
Access control	List of	--	Per product: Description(s) for access control
Terms-of-Use	products and	--	Per product: ToUCategoryID
Pricing & Ordering	productID	--	Per product: free or fee
Other			

Table 7: Description elements for Getcapabilities operation

9.3.2 Operation: DescribeProduct (productIDs)

Because product descriptions are expected to cover many different kinds of elements, e.g. for Access Control or ToU, the capabilities document should be unburned. Detailed product information can be obtained with the operation describedProduct.

In some cases also the relationship between products is important. If some parts of the business transaction may be specified by the user (or its client), e.g. time of license, the parameter and its list of potential values should be returned to the client in this step for the expected negotiation process. Table 8 shows potential elements.

DescribeProduct	general	Request	Response
Access control		--	--
Terms-of-Use	productID	--	Returns a text/schema with legal terms of use. Enhanced: Returns also potential choices for the negotiation phase
Pricing & Ordering			Returns the pricing model with potential choices
other			

Table 8: Description elements for DescribeProduct operation

9.3.3 Operation: NegotiatePreConditions (productIDs,conditions, UserID?)

A negotiation operation helps to optimize the conditions without any binding results. A user can set concrete values for configuration parameters and the service processes them and delivers a not binding offer. In some cases the userID may be an important factor for the result (see figure 24, path 6). Examples are updates or upgrades of existing business contracts. Table 9 shows potential elements.

NegotiatePreConditions		general	Request	Response
Access control			Optional: AC credentials and UserID	
Terms-of-Use		productID	TermsID, TermsParameter Selected Parametervvalues	Returns input depending text/schema with legal terms, e.g. for private use or within 1 year
Pricing & Ordering			PriceParameter, Selected Parametervvalues	Returns the prices
other				

Table 9: Description elements for NegotiatePreConditions operation

9.3.4 Operation: AgreePreConditions (productIDs,conditions, UserID)

The agreement is a (legally) binding operation. A user agrees to the negotiated conditions by an explicit interaction. In most cases the user should be identified for this operation. The result of the agreement is the right, often expressed as a license. The operation returns a token for the delivery operation. Table 10 shows potential elements.

AgreePreConditions		general	Request	Response
Access control			AC credentials and UserID	--
Terms-of-Use		ProductID	TermsID, TermsParameter, Selected Parametervvalues	Returns the license number
Pricing & Ordering			PricingParameter, Selected Parametervvalues	Returns a receipt with all pricing elements, tax, user data, etc. and a delivery token
other				

Table 10: Description elements for AgreePreConditions operation

9.3.5 Operation: DeliverProduct (token)

The last phase is the delivery. In some cases the delivery is carried out instantly after the agreement (order). In this case the operation AgreePreConditions may contain a flag to indicate that the response should already contain the product. In other cases a user may have the right to download actual data e.g. unlimited or within a year.

The separation between agreement and delivery is also useful for an asynchronous delivery, e.g. for large data files and for classic paper delivery.

Table 11 shows potential elements.

DeliverProduct		general	Request	Response
Access control			AC credentials and UserID	--
Terms-of-Use		ProductID	license number	--
Pricing & Ordering			Delivery token	
other				

Table 11: Description elements for DeliverProduct operation

The operations carry information elements for the business functions. SOAP already supports header/body transport. OASIS WS-S uses this approach, which was described and implemented in OWS3. Figure 30 shows a similar encoding approach. The transport of additional information elements is also possible to some extent with HTTP GET and HTTP POST.

Although mentioned above, it is expected that there are relationship between business function. If the relationships are dependencies only in a single direction the processing can be ordered. An example case is if the Terms-of-Use are depending on the user group. Another example case appears with pricing, if a price depends on a user, e.g. with special rebates.

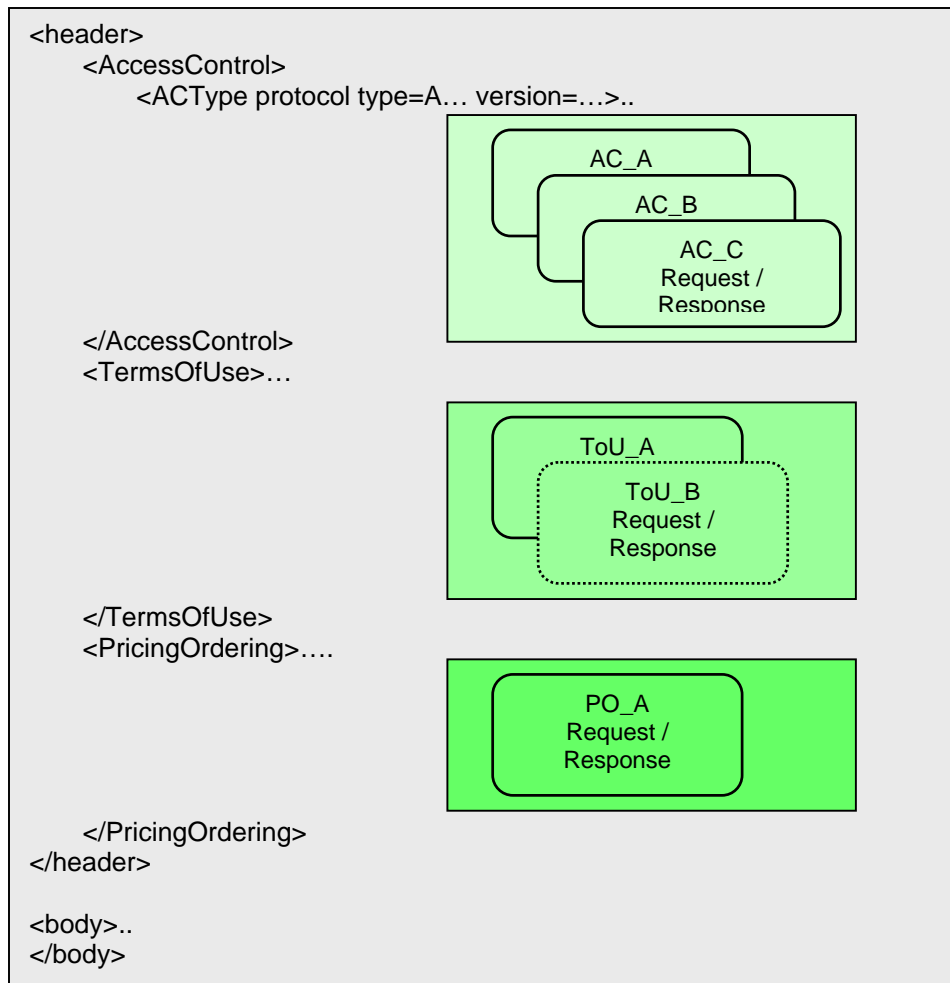


Figure 30: Encoding Approach

10 Outlook

OWS3 identified many business related issues. The carried out implementations delivered potential solutions. Although OWS3 focused on a simple click-through Terms-of-Use mechanism, it is obviously that other business functions, e.g. pricing & ordering are related. On the other hand it seems that a general description along the shown phases is possible, which may reduce much overlap and repetition.

The first step is to validate the identified structures (phases, business functionalities, descriptions, operations) and to refine them.

The second step is to develop the workflow description more in detail. Although chaining and workflow description languages might be helpful, a simple, but robust description seems to be sufficient.

A third step is the detailed population of information elements for each business function and the linkage with production description (ISO 19115 Metadata) and catalogues (CSW 2.0, profiles).

A separate and enhanced step might be a more detailed delivery phase with DRM support for a B2C market.

Bibliography

[1] deeJUMP with OWS-3 click-through:

<<http://wfs.lat-lon.de/deegree2/deejump.zip>>

[2] GDI NRW Web Authentication & Authorization Service v. 1.0:

<<http://www.gdi-nrw.org/iagent/upload/pdf/20030307094115.pdf>>

[3] GDI NRW Web Security Service v. 1.0:

<<http://www.gdi-nrw.org/iagent/upload/pdf/20030307094229.pdf>>

[4] deegree owsProxy:

<http://www.lat-lon.de/latlon/portal/media-type/html/language/en/user/anon/page/default.psml/js_pane/produkte%2Csub_produkte_deegree-igeosec>

[5] UniBW OWS3 SOAP Implementation, online at:

<<http://iisdemo.informatik.unibw-muenchen.de/ows3demo>>

[6] WS-Security, Version 1.0, online at:

<<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>>

[7] WS-Security Username Token Profile, Version 1.0, online at:

<<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>>

[8] Cristian OPINCARU, “Preliminary Study: Securing OpenGIS Web Services”, Runder Tisch GIS e.V. internal paper, January 2005

DACS_ACS DTD

```
<!ENTITY % common_decls SYSTEM "common.dtd">
<!ELEMENT dacs_acs (access_denied | access_granted | common_status)>
%common_decls;
<!ELEMENT access_denied (event900 | event901 | event902 | event903
| event904 | event905 | event998)>
<!ELEMENT event900 EMPTY>
<!ATTLIST event900
  handler CDATA #IMPLIED
  message CDATA #REQUIRED
>
<!ELEMENT event901 EMPTY>
<!ATTLIST event901
  handler CDATA #IMPLIED
  message CDATA #REQUIRED
>
<!ELEMENT event902 EMPTY>
<!ATTLIST event902
  handler CDATA #IMPLIED
  message CDATA #REQUIRED
>
<!ELEMENT event903 EMPTY>
<!ATTLIST event903
  handler CDATA #IMPLIED
  message CDATA #REQUIRED
>
<!ELEMENT event904 EMPTY>
<!ATTLIST event904
  handler CDATA #IMPLIED
  message CDATA #REQUIRED
>
<!ELEMENT event905 (notices)>
<!ATTLIST event905
  presentation_handler CDATA #IMPLIED
  ack_handler CDATA #IMPLIED
  notice_uri CDATA #REQUIRED
  resource_uri CDATA #REQUIRED
  time CDATA #IMPLIED
  hmac CDATA #IMPLIED
>
<!ELEMENT notices (notice_uri)+>
<!ELEMENT notice_uri EMPTY>
<!ATTLIST notice_uri
  uri CDATA #REQUIRED
>
<!ELEMENT event998 EMPTY>
<!ATTLIST event998
  handler CDATA #IMPLIED
```



```
    message CDATA #REQUIRED
>
<!ELEMENT access_granted EMPTY>
```

DACS_NOTICES DTD

```
<!ENTITY % common_decls SYSTEM "common.dtd">
<!ELEMENT dacs_notices (presentation_reply | ack_reply | common_status)
>
<!ELEMENT presentation_reply (notice)+ >
<!ATTLIST presentation_reply
  notice_uris CDATA #REQUIRED
  resource_uris CDATA #REQUIRED
  ack_handler CDATA #REQUIRED
  hmac CDATA #IMPLIED
  time CDATA #IMPLIED
>
<!ELEMENT notice (#PCDATA )>
<!ATTLIST notice
  uri CDATA #REQUIRED
>
<!ELEMENT ack_reply EMPTY >
<!ATTLIST ack_reply
  response (accepted | declined) #REQUIRED
  redirect CDATA #IMPLIED
>
```

DACS Common DTD

```
<!ELEMENT common_status EMPTY>
<!ATTLIST common_status
  context  CDATA #REQUIRED
  code     CDATA #REQUIRED
  message  CDATA #REQUIRED
>
```

CubeWerx NegotiateLicenses XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.opengis.net/wms"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:wms="http://www.opengis.net/wms"
  elementFormDefault="qualified">

  <element name="NegotiateLicenses">
    <complexType>
      <sequence>
        <element ref="wms:ForRequests"/>
        <element name="Accepted" type="string" minOccurs="0"/>
        <element ref="wms:JustAccepted" minOccurs="0"/>
        <element name="Format" type="string" minOccurs="0"/>
      </sequence>
      <attribute name="version" type="string" use="required"/>
      <attribute name="service" type="wms:OWSType" use="required"/>
    </complexType>
  </element>

  <element name="ForRequests">
    <complexType>
      <sequence>
        <element name="RequestUrl" type="string"
          minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>

  <element name="JustAccepted">
    <complexType>
      <sequence>
        <element name="LicenseId" type="string"
          minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
```


Terms of Use XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by
Westfälische Wilhelms-Universität Münster (Westfälische Wilhelms-
Universität Münster) -->
<schema xmlns:ows="http://www.opengeospatial.net/ows"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xlink="http://www.w3.org/1999/xlink"
targetNamespace="http://www.opengeospatial.net/ows"
elementFormDefault="qualified" version="0.1.4" xml:lang="en">
  <annotation>
    <appinfo>owsGetCapabilitiesResponse.xsd 2004/1/11</appinfo>
    <documentation>
      <description>This XML Schema encodes the GetCapabilities
operation response, also known as the Capabilities XML document. This
XML Schema must be expanded or edited by each OWS, to specify specific
contents of the Contents element and perhaps of the OperationsMetadata
element. </description>
      <copyright>Copyright (c) 2004 OpenGIS, All Rights Reserved.
</copyright>
    </documentation>
  </annotation>
  <!-- =====
and imports
===== -->
  <include schemaLocation="owsServiceIdentification.xsd"/>
  <include schemaLocation="owsServiceProvider.xsd"/>
  <include schemaLocation="owsOperationsMetadataAddedSoap.xsd"/>
  <include schemaLocation="owsGetCapabilitiesSubset.xsd"/>
  <!--
***** -
-->
  <!--Capabilities Request/Response-->
  <element name="GetCapabilities" type="ows:GetCapabilitiesType">
    <annotation>
      <documentation>Request</documentation>
    </annotation>
  </element>
  <element name="GeoDRMCapabilities">
    <annotation>
      <documentation>Response</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="ows:CapabilitiesType">
          <sequence>
            <element ref="ows:ProductCatalog"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

```

    <!--
*****
->
    <!--
*****
->
    <!--Negotiate Preconditions Request/Response-->
    <element name="NegotiatePreConditionsRequest">
        <annotation>
            <documentation>Request</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element name="ProductCatalog">
                    <complexType>
                        <sequence>
                            <element name="Product "
type="ows:ProductSubsetType"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
    <element name="NegotiatePreConditionsResponse">
        <annotation>
            <documentation>Response</documentation>
        </annotation>
        <complexType>
            <complexContent>
                <extension base="ows:CapabilitesType">
                    <sequence>
                        <element ref="ows:ProductCatalog"/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>
    <!--
*****
->
    <!--
*****
->
    <!-- ===== -->
    <complexType name="CapabilitesType">
        <annotation>
            <documentation>XML encoded GetCapabilities operation
response. This document provides clients with service metadata about a
specific service instance, usually including metadata about the
tightly-coupled data served. If the server does not implement the
updateSequence parameter, the server shall always return the complete
Capabilities document, without the updateSequence parameter. When the
server implements the updateSequence parameter and the GetCapabilities
operation request included the updateSequence parameter with the
current value, the server shall return this element with only the
"version" and "updateSequence" attributes. Otherwise, all optional

```

elements shall be included or not depending on the actual value of the Contents parameter in the GetCapabilities operation request.

```

</documentation>
  </annotation>
  <sequence>
    <element ref="ows:ServiceIdentification" minOccurs="0"/>
    <element ref="ows:ServiceProvider" minOccurs="0"/>
    <element ref="ows:OperationsMetadata" minOccurs="0"/>
  </sequence>
  <attribute name="version" type="ows:VersionType" use="required"/>
  <attribute name="updateSequence" type="ows:UpdateSequenceType"
use="optional"/>
</complexType>
<!-- ===== -->
<element name="OnlineResource" type="ows:OnlineResourceType">
  <annotation>
    <documentation>On-line information that can be used to
contact the individual or organization. OWS specifics: The xlink:href
attribute in the xlink:simpleLink attribute group shall be used to
reference this resource. Whenever practical, the xlink:href attribute
with type anyURI should be a URL from which more contact information
can be electronically retrieved. The xlink:title attribute with type
"string" can be used to name this set of information. The other
attributes in the xlink:simpleLink attribute group should not be used.
</documentation>
  </annotation>
</element>
<element name="WorkflowOfOperations"
type="ows:WorkflowOfOperationsType"/>
  <complexType name="WorkflowOfOperationsType">
    <sequence>
      <element name="OperationName" type="string"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="ProtocolName"/>
    <attribute name="ProtocolType"/>
    <attribute name="ProtocolVersion"/>
  </complexType>
<!-- =====
GeoDRM Functions-->
  <element name="Authentication">
    <complexType>
      <sequence>
        <element ref="ows:WorkflowOfOperations"/>
      </sequence>
    </complexType>
  </element>
  <element name="TermsManagement">
    <complexType>
      <complexContent>
        <extension base="ows:TermsManagementSubSetType"/>
      </complexContent>
    </complexType>
  </element>
  <element name="PricingAndOrdering">
    <complexType>
      <sequence>

```



```

        <element ref="ows:WorkflowOfOperations"/>
    </sequence>
</complexType>
</element>
<!-- =====
GeoDRM Functions-->
    <element name="ProductCatalog">
        <annotation>
            <documentation>The Tag ProductCatalog is the root element for
the producttree. This element helps to structure mulitple catalogs from
different vendors</documentation>
        </annotation>
        <complexType>
            <sequence>
                <element name="Product" type="ows:ProductType"
maxOccurs="unbounded">
                    <annotation>
                        <documentation>The business expression "product"
helps to abstract concrete layers, features, fids, services. A product
is a defined offer from a business point of view and has (business)
terms.</documentation>
                    </annotation>
                </element>
            </sequence>
            <attribute name="ProductCatalogID"/>
        </complexType>
    </element>
    <element name="Product" type="ows:ProductSubsetType"/>
    <complexType name="ProductType">
        <sequence>
            <element ref="ows:Title"/>
            <element ref="ows:Abstract"/>
            <element name="GeoDRMPreConditions">
                <complexType>
                    <sequence>
                        <element ref="ows:Authentication" minOccurs="0"
maxOccurs="unbounded"/>
                        <element name="TermsManagement"
type="ows:TermsManagementType" minOccurs="0" maxOccurs="unbounded"/>
                        <element ref="ows:PricingAndOrdering" minOccurs="0"
maxOccurs="unbounded"/>
                    </sequence>
                </complexType>
            </element>
            <element name="Resource" minOccurs="0">
                <annotation>
                    <documentation>A product is a business definition. It
is derived from resources, e.g. services or data.</documentation>
                </annotation>
                <complexType>
                    <sequence>
                        <element name="ResourceRecord"
maxOccurs="unbounded">
                            <annotation>
                                <documentation>A product definition may
consist of multiple ResourceRecords, e.g. multiple services. I a simple
case it is just a WMS layer.</documentation>

```

```

        </annotation>
        <complexType>
          <sequence>
            <element name="ResourceCapabilities"
minOccurs="0">
              <annotation>
                <documentation>A product resource
definition (OGC OWS common Capabilities) might be embedded in the
product definition or referenced. In the case that a product is
referenced within the XML document, the ResourceCapabilities Element is
optional.</documentation>
              </annotation>
              <complexType>
                <sequence>
                  <element
name="EmbeddedCapabilities" minOccurs="0">
                    <annotation>
                      <documentation>In the
case of embedded OGC OWS common capabilities, the complete XML document
is stored in the GeoDRM Capabilities Document. In the case of a
reference only the URL need to be defined in the
OnlineResourceblock</documentation>
                    </annotation>
                    <complexType>
                      <sequence/>
                    </complexType>
                  </element>
                  <element
ref="ows:OnlineResource" minOccurs="0"/>
                </sequence>
              </complexType>
            </element>
            <element name="ResourceType"
type="string" minOccurs="0">
              <annotation>
                <documentation>XML Tag name, e.g.
layer, fid</documentation>
              </annotation>
            </element>
            <element name="ResourceId" type="string"
minOccurs="0" maxOccurs="unbounded">
              <annotation>
                <documentation>XML Tagfield Value,
e.g. water, "123"</documentation>
              </annotation>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
<element name="Product" type="ows:ProductType" minOccurs="0"
maxOccurs="unbounded"/>
  <element name="TAN" minOccurs="0" maxOccurs="unbounded">
    <annotation>

```

```

        <documentation>Some Business Models have higher trust
requirements. Therefore TransactionNumbers (TAN) can be used for a
higher level. The mechanism is often used for home banking. A user
might acquire the TANs by various ways. A defined ways are: 1.)
Publishing in the Capabilities document as a static or dynamic value,
2.) delivery by GeoDRM response</documentation>
        </annotation>
    </element>
</sequence>
<attribute name="ProductId" use="required">
    <annotation>
        <documentation>a productid is an ID which represents a
productdescription</documentation>
    </annotation>
</attribute>
</complexType>
<complexType name="ProductSubsetType">
    <sequence>
        <element ref="ows:Product" minOccurs="0"
maxOccurs="unbounded" />
        <element name="TAN" minOccurs="0" maxOccurs="unbounded" />
        <element ref="ows:GeoDRMPreConditions" minOccurs="0" />
    </sequence>
    <attribute name="ProductId" use="required" />
</complexType>
<element name="GeoDRMPreConditions">
    <annotation>
        <documentation>Some products may require some preconditions
to be handled first. Examples are Authentication or the acknoweldgement
of terms. This element shows, which GeoDRM Functions are required and
who to access them (by operations and URI)</documentation>
    </annotation>
    <complexType>
        <sequence>
            <element ref="ows:Authentication" minOccurs="0" />
            <element name="TermsManagement" minOccurs="0">
                <complexType>
                    <complexContent>
                        <extension
base="ows:TermsManagementSubSetType" />
                    </complexContent>
                </complexType>
            </element>
            <element ref="ows:PricingAndOrdering" minOccurs="0"
maxOccurs="unbounded" />
        </sequence>
    </complexType>
</element>
<complexType name="TermsManagementType">
    <sequence>
        <element name="terms">
            <complexType>
                <choice>
                    <element name="EmbeddedTerms"
maxOccurs="unbounded" />
                </choice>
            </complexType>
        </element>
    </sequence>
</complexType>

```

```

        <extension base="string">
            <attribute name="language">
                <annotation>
                    <documentation>Identifier of
the language used by all included exception text values. These language
identifiers shall be as specified in IETF RFC 1766. When this attribute
is omitted, the language used is not identified. </documentation>
                </annotation>
            </attribute>
        </extension>
    </simpleContent>
</complexType>
</element>
    <element ref="ows:OnlineResource"/>
</choice>
    <attribute name="termsId" use="required"/>
</complexType>
</element>
    <element ref="ows:WorkflowOfOperations"/>
</sequence>
</complexType>
<complexType name="TermsManagementSubSetType">
    <sequence>
        <element name="terms">
            <complexType>
                <attribute name="termsId" use="required"/>
            </complexType>
        </element>
    </sequence>
</complexType>
</schema>

```