XCLX: An XML-based Common Logic eXtension with Embedded Geography Markup Language

Mary E. Athan

September 2011

A dissertation submitted in partial fulfillment of the requirements of the Masters Degree in Geographic Information Systems of the University of Leeds

Abstract

A novel eXtensible Markup Language (XML)-based Common Logic eXtension, called XCLX, is presented. The novel syntax draws from the standard syntaxes Common Logic Interchange Format (CLIF) and eXtended Common Logic Markup Language (XCL), as well as Rule Markup Language (RuleML), Interoperable Knowledge representation Language (IKL), IKRIS Context Language (ICL), and XML Inclusions (XInclude). In addition, the syntax is open to user extensions, including embedding elements from foreign namespaces as names, functions and atomic sentences, and has a meta-language for self-extensibility. These features allow Common Logic to embed structured data, such as Geography Markup Language (GML), while maintaining a well-defined semantics. The overall syntax is defined by a modular schema, using a design pattern developed for RuleML. The XCLX semantics is defined either by formal mappings into equivalent syntactic forms (in XCLX or foreign dialects, including CLIF, IKL, ICL), or axiomatically through the meta-language. The semantics of the meta-language is novel and is stated directly. A number of illustrative examples, drawn from geography and GML, are presented.

Appendix 1: Declaration of Academic Integrity

UNIVERSITY OF LEEDS

SCHOOL OF GEOGRAPHY
UNIVERSITY OF LEEDS

DECLARATION OF ACADEMIC INTEGRITY

This form must be completed, signed and attached to every dissertation submitted for a degree in Geography at the University of Leeds.

I have read the University regulations on cheating and plagiarism and I confirm that this piece of work is my own and does not include any unacknowledged work from any other sources.

I understand that the title of this dissertation and my name, as its author, may be included in an on-line catalogue of School of Geography dissertations. Also, that the mark/grade may be recorded on the dissertation when it is stored in the School of Geography. I agree to this.

Signed..... Date.....2011-09-11.....

Surname	Athan
First names	Mary

Programme of Study

Postgraduate	MA		MSc	X

Masters Programme Title: Geographic Information Systems

Other *(give programme title)*:

Dissertation title

XCLX: An XML-based Common Logic eXtension with Embedded Geography Markup Language

Dissertation tutor: Oliver Duke-Williams

Word length*: 14845

** Word length - exclude bibliography, appendix, tables, diagrams*

Acknowledgments

I wish to thank RuleML, Inc. for partial support of the work presented in this dissertation, and Harold Boley for many fruitful discussions, the Common Logic community, including John Sowa, Chris Menzel and Pat Hayes, for help in understanding the Common Logic and IKL specifications, Bill Andersen for use of the Common Logic Sourceforge repository, my adviser Oliver Duke-Williams for advice in the preparation of the dissertation, my parents for their unwavering support, my children Penelope and Helen, for understanding, and my husband, Jason, for being always by my side, as you will always be in my heart.

Table of Contents

Index of Tables

Index of Code Blocks

Illustration Index

Chapter 1 Introduction

Section 1.1 Background and Motivation

Logicians devised notation systems as aids to reasoning long before the computer age (Boole 1854), but with the advent of electronic computers, machine-readable logical notation systems became the focus of attention, with Lisp, introduced in 1958, being the first such language to have a significant longevity (McCarthy 1979). The underlying abstract structure of Lisp is the "list", whose concrete syntax encloses a sequence of whitespace-separated elements in parentheses; nearly everything, from data to programs, is considered a list, and these lists may be nested to form complex structures that are equivalent to the data structure called "trees" in computer science.

Code Block 1.1: Lisp as list data for y = x + 1

```
(= y (+ x 1))
```

The tree equivalent to the Lisp code in Code Block 1.1 is shown in Illustration 1.

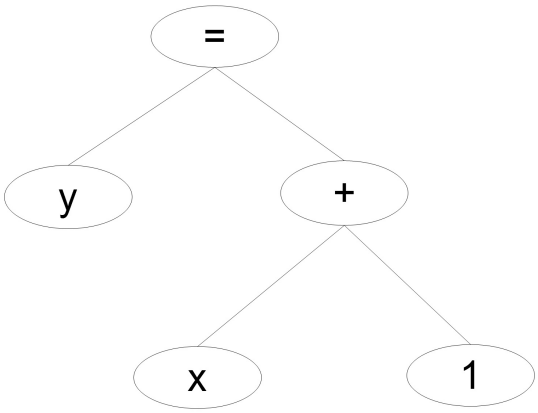

Illustration 1: Tree diagram of Lisp equivalent of y = x + 1.

Based on Lisp and similar languages, electronic storage, processing and retrieval of knowledge has been implemented on a wide variety of systems using numerous representations. In the 1990's, KIF (Knowledge Interchange Format) was developed "for use in the interchange of knowledge among disparate computer systems " (Genesereth

1998). KIF was strongly influenced by Lisp, and so uses a similar style of notation, with parentheses and whitespace for delimiters, and prefix operators (the operator is always the first item in a list). One of the larger knowledge representation (KR) projects is the Suggested Upper Merged Ontology (SUMO), written in the KIF variant SUO-KIF (Pease 2009). Code Block 1.2 gives an excerpt from SUMO including structural axioms for a class of objects representing geographic coordinates.

Code Block 1.2: A SUMO excerpt in SUO-KIF courtesy of (Pease 2009).

```
(subclass Latitude Region)
(subclass Longitude Region)
(instance objectGeographicCoordinates TernaryPredicate)
(domain objectGeographicCoordinates 1 Object)
(domain objectGeographicCoordinates 2 Latitude)
(domain objectGeographicCoordinates 3 Longitude)
```

In parallel to these developments, formats were being developed for storing, processing and retrieval of numerical data. At about the same time as the development of KIF, a need was recognized for a format for exchanging structured data between disparate computer systems, leading to the development of XML (W3C 2008). Like Lisp and its descendants, the abstract structures of XML documents are trees. The XML language has a different style than the languages on the Lisp branch, using angle brackets '< >' but not whitespace as delimiters of tags, and tags themselves as delimiters of elements. The equivalent to Code Block 1.1 is shown in Code Block 1.3 in the XML-based MathML (W3C 2010).

Code Block 1.3: MathML for y = x + 1

```
<apply>
  <eq/>
  <ci>y</ci>
  <apply>
    <plus/>
    <ci>x</ci>
    <cn>1</cn>
  </apply>
</apply>
```

The tree equivalent to the MathML in Code Block 1.2 is shown in Illustration 2. Notice that although we are representing the same concept as in Illustration 1, the tree is

different.

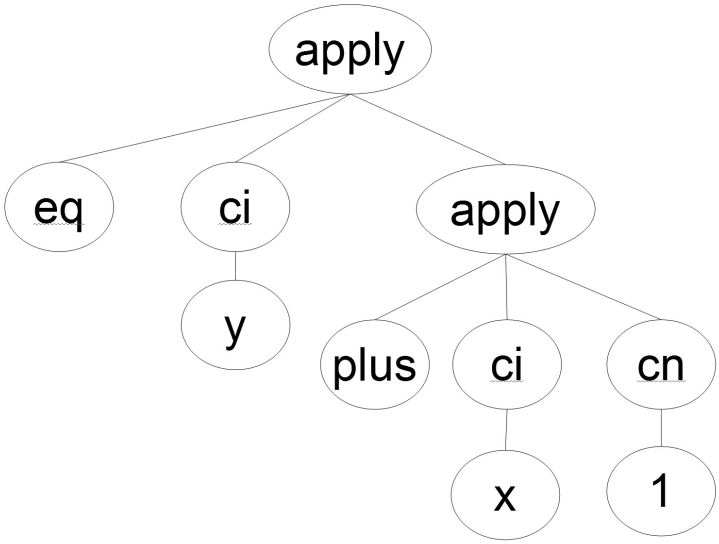

Illustration 2: Tree diagram of the MathML equivalent to $y = x + 1$.

XML gained popularity as its extensions, including the extensible HyperText Markup Language (xHTML) (W3C 2000) and Resource Description Framework (RDF) (Beckett and McBride 2004), became workhorses of the World-Wide Web. Among other domain-specific extensions, an XML-based language for geography (GML) was developed (OGC 2007b), and a significant amount of data with geospatial content is available in this format through applications such as Geoserver (Geoserver 2010). In particular the Web Feature Service (WFS) has an option to request data in the GML format (OGC 2010b). Another XML-based format widely used in geography is KML (OGC 2007a), which is oriented towards visual markup and used in mapping applications such as Google Earth (Google 2011).

The structured data example contained in Code Block 1.4 is an instance of a sample GML application schema, published in (OGC 2011), which uses the GML simple features profile. This instance contains knowledge about a news item, including spatial location and other non-spatial attributes, such as the associated reporter.

The one of the first KR language to be implemented as an XML extension was the Ontology Interchange Language (OIL) (Decker *et al.* 2000), which has been superseded by Web Ontology Language (OWL) (W3C 2004a) and OWL2 (W3C OWL Working

XLCX and GML

Code Block 1.4: XML instance of a GML 3.2 application schema.

```
1  <?xml-model href="newsitem_script.nvdl"
2    type="application/xml"
3    schematypens="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0"?>
4  <gml:FeatureCollection xmlns:gml="http://www.opengis.net/gml/3.2"
5    xmlns:nws="http://www.cubewerx.com/cw" gml:id="G1">
6    <gml:featureMember>
7      <nws:Reporter gml:id="G2">
8        <nws:reporterId>R1</nws:reporterId>
9        <nws:email>R1@newscorp.com</nws:email>
10     </nws:Reporter>
11     <nws:NewsItem gml:id="G3">
12       <nws:location>
13         <gml:Point gml:id="G4" srsDimension="2"
14                    srsName="urn:ogc:def:crs:EPSG:6.6:4326" >
15           <gml:pos>29.5 -123.05</gml:pos>
16         </gml:Point>
17       </nws:location>
18       <nws:reporterId>R1</nws:reporterId>
19       <nws:eventDate>2011-08-16T14:05:38</nws:eventDate>
20       <nws:byLine>Tara Athan</nws:byLine>
21       <nws:details>Blah blah.</nws:details>
22       <nws:image mimeType="image/jpeg"
23             url="http://www.mynewsservice.org/jpeg-ar/324589.jpg"/>
24     </nws:NewsItem>
25   </gml:featureMember>
26 </gml:FeatureCollection>
```

Group 2009). Other XML-based KR languages include RuleML (Boley 2003b), which began as a rules language, but has since been extended to full first-order logic and beyond. RDF itself became a KR language, albeit of limited expressive power, when it was given a model-theoretic semantics (meaning) (Hayes 2004).

The latest attempt to bring interoperability to this diverse and dynamic situation is the Common Logic (CL) project, which resulted in an international standard (ISO/IEC 2007). This standard includes an abstract syntax and model-theoretic semantics, which are to be followed, to a greater or lesser degree according to various levels of conformity, by any language claiming to be a CL dialect. Further, the standard published three dialects, the Lisp-style Common Logic Interchange Format (CLIF), the XML-based eXtended Common Logic Markup Language (XCL), and the Conceptual Graph Interchange Format (CGIF), which has a different style altogether.

The following Code Block 1.5 contains a CLIF sentence equivalent to lines [13-16] of Code Block 1.4.

Code Block 1.5: A CLIF sample.

```
(exists (x
"{http://www.opengis.net/gml/3.2}Point?
srsDimension=2&amp;srsName=urn:ogc:def:crs:EPSG:6.6:4326")
  (and
    ("{http://www.opengis.net/gml/3.2}pos"
      ({http://www.w3.org/2001/XMLSchema}double '29.5')
      ({http://www.w3.org/2001/XMLSchema}double '-123.05'))
    (exists (y "{http://www.cubewerx.com/cw}NewsItem")
       ("{http://www.cubewerx.com/cw}location" y x)
) ) )
```

However there is still a gap in interoperability between structured data formats, such as GML, and KR languages. Much of the semantics of geospatial data in the GML format is contained in conceptual models, application schemas and external metadata. For example, the information that the contents of the **<gml:pos/>** element (line number [15] in Code Block 1.4) is a list of double precision floating point numbers is not evident in this instance document; that information is contained in the GML schema, which is referenced indirectly from the processing instruction on line [1].

While such external storage of semantic information makes the data format more efficient, it poses a challenge for machine-processing of this knowledge by domain-neutral applications that have not internalized the GML semantics. An earlier study on incorporating geospatial information into RuleML utilized a non-standard geometry notation and added geospatial Java libraries to the reasoner (Gao *et al.* 2009). Efforts to bridge the gap from the other direction include (Fonseca and Llano 2011), where standard GIS formats are used, but a novel method of representation, Data-Representation Ontology, is proposed. Here we explore a different approach; the primary aim of this dissertation is to present a method for directly expressing the full semantics of a standard geospatial data format, GML, in a general-purpose KR language, based on existing KR language syntax and/or semantics to the extent possible.

For our purposes, an XML-based syntax is preferred, in order to allow GML to be directly embedded. There are also a number of "freebies" that come with using an XML-based language, including qualified names, datatypes and relative references (see Section 3.2). Note in Code Block 1.5 there is repetition of the namespace

"**`http://www.w3.org/2001/XMLSchema`**"; XML allows the substitution of a short prefix for a long namespace, while CLIF has no built-in namespace capability.

We evaluated the available XML-based KR languages for suitability in this project. Unfortunately, the XCL dialect has some flaws, as detailed in (International Organization for Standardization 2008) and so is not currently usable. Also the design pattern of the XCL syntax is not very amenable to extension in the manner needed for our purposes. Other XML-based KR languages, such as RDF and OWL, have limited expressive power, which makes it impossible to express some of the axioms needed to accomplish our goals. RuleML has the greatest expressive power, but the semantics for its entire family of languages has not been formally specified, and its self-extensibility[1] is limited (Boley 2003a). Therefore, we have developed a novel XML-based syntax called XCLX, while adopting and extending the CL model-theoretic semantics to include sentences and terms in the form of embedded structural data and foreign dialects, contextual statements, and a built-in meta-language that enables self-extensibility.

As illustration of sentences as embedded structural data and foreign dialects, the following Code Block 1.6 contains a statement translating a GML Geometry object (lines [4-8]) of type Point into a CLIF sentence (lines [10-17]).

1 Self-extensibility is the capability of a language to refer to and extend itself. Lisp macros are an example of self-extensibility.

Code Block 1.6: An XCLX sample of embedded structural data and a foreign dialect.

```
1  <xclx:Equiv xmlns:xclx="http://www.example.org/xclx/1"
2    xml:base="http://www.example.org/newsitem_instance1/">
3
4    <gml:Point xmlns:gml="http://www.opengis.net/gml/3.2"
5      gml:id="G4" srsDimension="2"
6      srsName="urn:ogc:def:crs:EPSG:6.6:4326">
7      <gml:pos>29.5 -123.05</gml:pos>
8    </gml:Point>
9
10   <xclx:sentence cdialect="#clif"> (exists (x
11     "{http://www.opengis.net/gml/3.2}Point?
12 srsDimension=2&amp;srsName=urn:ogc:def:crs:EPSG:6.6:4326")
13     (and (= x "http://www.example.org/newsitem_instance1#G4")
14     ("{http://www.opengis.net/gml/3.2}pos" x
15     ({http://www.w3.org/2001/XMLSchema}double '29.5')
16     ({http://www.w3.org/2001/XMLSchema}double '-123.05')) ) )
17     ) </xclx:sentence>
18
19 </xclx:Equiv>
```

Sentences such as Code Block 1.6 are just the beginning of what is needed to express the semantics of GML in general. First, it would be very inefficient to state such a sentence for every feature in a dataset. Instead, we would like to assert that such a sentence is true not just for the numerical values of 29.5 and -123.05, but for all appropriate numerical values. Further, we would want to have such a statement for every spatial reference system (SRS). However the bounding rectangle of appropriate coordinates is different for different SRSs (especially projected SRSs), so we need a statement that takes a structured data specification of an SRS as a premise, and generates as a conclusion an equivalence between a general GML point definition and a corresponding sentence in a KR language. Such assertions may be made in the XCLX meta-language (see Code Block 10.2 for an example).

The statements in Code Block 1.6 refer to abstract concepts about geometry and spatial reference systems that are fixed by convention and do not change in time. In contrast, data in reference to observations of real world phenomena tend to have a finite time interval of validity. This is illustrated by the statements in Code Block 1.4, which make claims about the spatial position associated with some newsworthy event, which will certainly have some temporal interval of validity that is not stated explicitly in the document.

XLCX and GML

In general, we collect data because we believe it tells us something about the real world. We can perform exact reasoning such as Code Block 1.5 as long as we limit our universe of discourse to abstract entities, but as soon as we shift our focus to the real world objects of observation, any statement of knowledge must be put into context. For example, in Code Block 1.7, we show information extracted from Code Block 1.4 based on the location of the news event.

Code Block 1.7: An XCLX contextual statement.

```
1  <Atom> <op><Var cri="/cxt_1#default"/></op>
2    <That>
3      <Atom> <op> <nws:location/> </op>
4        <Var cri="#R1"/>
5        <Var cri="#G4"/>
6      </Atom>
7    </That>
8  </Atom>
```

The nested atomic sentence (lines [3-6] in Code Block 1.7) asserts that the object with identifier "`#R1`" (in this case, a reporter) was at the location with identifier "`#G4`". The top-level atomic sentence, whose delimiters are in lines [1, 8] provides an identifier, in this case the relative reference "**`/cxt_1#default`**", for the context of this assertion.

When specifying the properties of the context "**`/cxt_1#default`**", multiple facets should be considered, including but not limited to:

- temporal dynamics, in that the dataset does not support the assertion about the reporter's location at any time other than the time period associated with the news event;
- degree of belief; in many cases, one would expect the reporter to be on location to observe the event, but there are also situations where the reporter may have arrived after the event began, or have never been to the location but is reporting based on interviews of other observers; further we may have limited faith in the reliability of the original data;
- spatial uncertainty; one does not expect the geographic coordinates to describe the reporter's location precisely, but only approximately;

- authority, including the publisher and the lineage (the supporting data and the method of inference).

Such implicit contextual inference is commonplace in the utilization of geographical data, whether it is meteorological data from satellites used as input into climate models, Global Positioning System (GPS) tracks from radio collars on wildlife used to develop species range predictions, regional employment statistics from a census used to guide government policy, or any other observation with geospatial components when used by a spatial analyst to draw conclusions. What is less common is the formal expression of the data and its context in a machine-readable format from which contextual knowledge may be automatically extracted.

Section 1.2 Goals

We state here the goals of this dissertation as motivated in the previous subsection:

1. Allow embedded structured data, such as GML features and Geometry elements, to be used as sentences, and their identifiers as names in a model-theoretic KR language with high expressive power and a solid semantic foundation;
2. Take advantage of XML strengths, including its capabilities for namespaces, datatyping and relative references, schemas for validation, stylesheets for transformation, etc.;
3. Enable self-extensibility of the language through a built-in meta-language that allows character-level rewriting of texts;
4. Set the stage for an XML-based contextual KR language, including contextual interpretation of the vocabulary.

Section 1.3 Objectives

1. Develop a novel XML-based CL extension, called XCLX, with the following properties:
    a. follows the serialization syntax of RuleML where possible;
    b. has a unified naming framework based on a single element (`<Name/>`) ;

c. has unified frameworks for Boolean and quantified sentences based on elements (**<Boolean/>**, **<Quantification/>**) with a modifying attribute (**@kind**);

d. is compatible with foreign namespaces, which may be integrated into the syntax or be directly embedded as structured data;

e. has a fully-striped[2] (Brickley 2004) form and a compact form where redundant stripes are optional;

f. has abbreviated elements (shortcuts) (**<Var/>**, **<Data/>**, **<That/>**, **<Forall/>**, ...) for the most frequently used patterns;

g. allows direct embedding of terms and sentences from other dialects, such as CLIF;

h. is self-extensible with an eXtensible Stylesheet Language Transformations (XSLT)-style (Clark 1999b) meta-language that enables quantification over structural components as well as arbitrary sub-strings of a text;

2. Formally define the semantics of XCLX, when possible through mappings to other KR languages (CLIF, IKL) or to equivalent XCLX forms;

3. Validate and illustrate the XCLX syntax and semantics with examples from geography.

Section 1.4 RoadMap

In Chapter 2, the literature relevant to this development of XCLX is reviewed. In Chapter 3, the abstract specifications of XCLX are developed based on a more detailed analysis of the capabilities of current languages. In Chapter 4, the **ab**breviated form of the XCLX language, XCLX-ab, is presented, including the syntax and semantics. An overview of XCLX extensions is given in Chapter 5. In Chapters 6-9, the unified frameworks for naming and several sentence categories are briefly described. In Chapter 10, the general approach for using XCLX to express the full semantics of GML instances is described. A summary of the current work and a discussion of open questions is

2 Striped XML is a style based on alternating nested tags, representing nodes and edges of a graph-based model. This style was first introduced by RDF, and has been adopted by RuleML and GML.

presented in Chapter 11. Electronic versions of the schemas and instance examples are on the disk submitted along with this dissertation[3].

3 Schemas and examples are also downloadable from "http://sourceforge.net/projects/common-logic/".

Chapter 2 Literature Review

This section contains a literature review of several KR and structured data languages as background for the development of XCLX, including Common Logic Interchange Format (CLIF), eXtended Common Logic Markup Language (XCL), Interoperable Knowledge representation Language (IKL), Rule Markup Language (RuleML), IKRIS Context Language (ICL) and Geography Markup Language (GML).

These languages are used in a range of fields, including business and commerce, decision support, and intelligence analysis, as formats for information exchange, and as direct input into machine reasoners.

We organize the review by themes. In Section 2.1 and Section 2.2, we compare the syntax and semantics, resp., of the languages at a general level. Additional details of the languages are discussed in Chapter 3, where the abstract specifications of XCLX are developed.

The normative reference for the CL abstract syntax and semantics, and the concrete CL dialects is the CL Standard (ISO/IEC 2007). Similarly, the IKL Specification (Hayes and Menzel 2006?) is normative for IKL, while the IKL Guide (Hayes 2006?) is informative. The normative definition of RuleML (Boley, Paschke and Shafiq 2010) was obtained from "`http://ruleml.org/spec/`", including (Hirtle, Dema and Boley 2006) as well as my own work carrying out the re-engineering of RuleML into Relax NG (Athan and Boley in press).

Sources for GML include the OpenGIS Abstract Specification (OGC 2009), the Open Geospatial Consortium (OGC) Reference Model (OGC 2010a), the OpenGIS GML Standard (OGC 2007b), and the GML Simple Features Profile (OGC 2011).

Section 2.1 Syntax

In this section we compare the syntax of the KR and structured data languages at a general level, including syntactic categories, and the approaches used to specify the syntax.

XLCX and GML

CL Abstract Syntax

The abstract syntax defined in the CL Standard (ISO/IEC 2007) identifies syntactic categories of expressions, including names, terms, term sequences and sequence markers, sentences, phrases, and texts. The vocabulary of a text is the set of all names and sequence markers. The details of the syntax for each category are discussed in Section 3.2 - Section 3.10 below. Further, the CL Standard recommends that a single XML syntax be used for the exchange of CL content.

Grammars for the Lisp-style Syntaxes

Extended Backus-Naur Format (EBNF) grammar is used to specify the CLIF syntax and IKL syntax, while ICL (IKRIS 2007a) uses the ANTLR notation.

RuleML Syntax

Originally the RuleML syntax was specified by a Document Type Definition (DTD) schema. In later versions, the syntax was converted to XML Schema Definition Language (XSD), and recently the syntax was re-engineered in Relax NG, for improved modularization and extensibility. Relax NG also has advantages over XSD in the ability to express non-deterministic patterns. It is particularly useful for representing content models that depend on the value of attributes. One drawback to Relax NG compared to XSD is the lack of maturity of software to generate parsers directly from Relax NG schemas. The Relax NG schemas of RuleML were designed so that they could be translated into XSD for the auto-generation of parsers.

GML Syntax: Conceptual Models

According to (World Meteorological Organization 2004), GML is based on a conceptual model approach, in which a conceptual schema is used to specify both syntax (structure) and semantics. Although any conceptual schema language, or its equivalent, is allowed by the GML Standards, the specifications use the Unified Modeling Language (UML) for the meta-model level schema, and XSD is assumed in the specifications for the application schema level.

Comparison of Syntax in Structured Data and KR languages

A conceptual schema restricts the classes of entities that may be referred to in a structured

data instance. Compared to the syntax of logical languages, this is quite limiting. In CL, for example, a new class or property is introduced simply by using a new name anywhere in a sentence, because every name in the vocabulary is required to denote a relation as well as an individual[4]. In contrast, in a conceptual schema, a new class or property may be introduced only by extending the schema - a class or property cannot be introduced in an instance document.

Section 2.2 Semantics

The syntax of a formal language provides requirements that must be followed for any instance of the language to be considered (syntactically) valid. An interpretation is what provides the meaning of valid instances, and, the semantics of a formal language provides, among other things, requirements that must be followed for a structure to be considered an interpretation.

CL Semantics

In the CL semantics, an interpretation of a vocabulary V is a structure that includes

1. a universe of reference UR_I
2. a universe of discourse UD_I
3. int_I - a mapping from the vocabulary V to the universe of reference UR_I
4. rel_I - a mapping from UR_I to sets of finite sequences of elements of the universe of discourse UD_I.
5. fun_I - a mapping from UR_I to total functions from finite sequences of elements of the universe of discourse UR_I into UR_I.
6. seq_I - a mapping from the vocabulary V to finite sequences of elements of the universe of discourse UD_I.

A text ***T*** is satisfied in an interpretation *I* if it is given a truth-value of *true* when *I*(***T***) is evaluated according to the semantics of the language. Additional details, provided in the

4 In an interpretation, the relational extension of a name **n** is the set of arguments that are mapped to the value *true* by the relation $rel_I(int_I(\mathbf{n}))$. The relational extension is some subset (possibly empty) of the set of finite sequences from the universe of discourse.

CL Standard (ISO/IEC 2007), that are relevant to this work are discussed in Section 3.1 - Section 3.11 below.

Some non-traditional aspects of these interpretations are:

1. A name may denote an individual, a relation and a function; the position of the name in an expression, i. e., its syntactic category, determines which mapping is applied during interpretation.

2. In general, functions and relations do not have a specified "arity". There is no such thing as binary functions or ternary predicates. A relation may take on a specific "arity" if its extension contains only sequences of a particular length. A function, however, is required to be "total"; it must have an image for sequences of all lengths, including the empty sequence.

3. Sequence markers cause Common Logic to exceed first-order logic. By quantifying over bound sequence markers, it is possible to make assertions concerning sequences of arbitrary length. This is equivalent to asserting a countably-infinite number of first-order logic sentences, but first-order logic applies only to finite collections of sentences.

The CL Standard (ISO/IEC 2007) also addresses the publication and exchange of CL content on a "network", such as the Internet. A particular challenging requirement from (ISO/IEC 2007) is

> "All texts which are published and identified on a network shall be mutually interpretable with all other texts on the network which can import them, over the same universe of reference and domain of discourse, and with their vocabularies merged."

The Standard also recommends that when implemented on a network, the meaning of names and texts, and the entailments they support should be independent of location on the network. These requirements limit indexical names, such as "I", "here", and "now", to be contained within explicit contexts (see Section 3.9)

Semantics of IKL

IKL is a dialect of CL, and so adopts, and extends, the model theory of CL. The main extension of the CL semantics implemented by IKL is in regard to named propositions, and is discussed in s Section 3.2 and Section 3.11 below.

Semantics of GML

The conceptual models at the "meta-model" level of GML are adopted from the OpenGIS Abstract Specification, and include the Reference Model and General Feature Model, specified in (OGC 2009). In ISO 19101, a feature is defined as "an abstraction of a real world phenomenon" ((ISO 2005) in (OGC 2009)); in other words, a feature is not itself a physical entity, it is an information object that is "about" some physical entity. This is a critical distinction, because, unlike real world objects, the knowledge about information objects is finite - it is possible to say everything there is to say about a "feature" in a finite character sequence. The General Feature Model semantics carries a "closed-world assumption" regarding its information objects, but allows reference to real (or possible) world objects, which requires an "open-world assumption". In a closed world, we assume that our knowledge is complete, while in an open world, we do not assume that just because a certain fact is not entailed by our knowledge base, it is not true (Reiter 1978).

A geographical feature has a geospatial extent, which may be represented by either vector or raster data in a geocentric spatial reference system. A geographical feature also has attributes describing "measurable or describable properties of this entity", from (Kottman and Reed 2009, p. 1) in reference to (ISO 2005).

Semantics of GML Application Schemas

In addition to the semantics at the meta-model level, additional semantic information is defined within a GML application schema. Derivation of object types define subclass relationships and the types of properties defines additional subclass relationships for their domain and range (see Section 3.8 for more details).

Comparison of the Semantics of Structured Data and KR Languages

Much of the semantics of structured data is stated informally, in natural language, and is often domain-specific. In comparison, the semantics of KR languages is domain-neutral.

XLCX and GML

Although expressed in formal language, it is usually published as human-readable specifications, which are then implemented in software applications. However, in the case when the language has self-extensible capabilities, e. g. IKL, some of the semantics may be stated axiomatically.

Chapter 3 Specifications of XCLX

The specifications of XCLX are motivated by the objective that the language be a conforming dialect of CL. Within the confines of these requirements, there is still considerable freedom. The objectives to allow embedded GML and to have self-extensibility further shape the specifications.

We organize the discussion according to the syntactic categories defined in the CL Standard (ISO/IEC 2007), and begin each section with a review of the approach taken by the various KR and structured data languages reviewed in Chapter 2. In Section 3.1 we discuss the lexical level of the languages, including character encoding and tokenization. In Section 3.2 - Section 3.4, we review the various syntactic categories of the languages at the terminological level, from character encoding to term sequences. The logical level is the focus in Section 3.5 - Section 3.8, while in Section 3.9, we compare the approaches to context that are available in several of the languages. In Section 3.10 - Section 3.11, we review the extra-logical features of the languages, including collections of sentences as texts, and self-extensibility.

Section 3.1 Encoding and Tokenization

The CL Standard (ISO/IEC 2007) recommends that names should be defined in terms of Unicode conventions, and this recommendation is followed by all languages considered here. The Lisp and XML-based languages differ on their methods of tokenization as discussed above; there are advantages and disadvantages to either approach. Our choice of an XML-based approach arises from the need for compatibility with GML rather than any preference in the serialization approach.

3.1.a XCLX Encoding and Tokenization Specifications

XCLX is specified to be XML-based, which enables satisfaction of the CL requirement for a Unicode-based encoding (UTF-8 and UTF-16), and completely specifies the tokenization approach, according to the XML specifications (W3C 2008).

Section 3.2 Names

This is the most complex syntactic category, where significant differences arise among all

the languages considered here.

Names in the CL Abstract Specifications

The CL Standard (ISO/IEC 2007) provides requirements that must be followed in order for a language to be considered a CL dialect, as well as some recommendations. Because of the crucial role that names play in the language, there are quite a few of these requirements and recommendations that affect our development of the syntax and semantics of names in XCLX.

Section 5, p. 7 of the CL Standard identifies several desired goals that pertain to names, especially with regard to the general goals that CL should "be easy and natural to use on the Web", and "support open networks". In particular, "URIs and URI references SHOULD be usable as names", "URIs should be usable to give names to expressions and sets of expressions" and "Every name should have the same logical meaning at every node of the network".

Since the CL Standard was published, the Uniform Resource Identifier (URI) syntax has been complemented by the Internationalized Resource Identifier (IRI) syntax. Although usage of IRIs is not an explicit requirement of the CL Standard, it is utilized in CLIF and so in the rest of the discussion IRIs will be used instead of URIs as the Internet standard for network identifiers.

The CL Abstract Syntax (ISO/IEC 2007) also imposes a few requirements on names.

- Item 6.1.1.14 requires that names and sequence markers (names for a finite sequence of terms) be disjoint.
- Item 6.1.1.10 states that the lexical category 'terms' includes names, (as well as functional terms and sequence markers), so the many expressions that make use of terms provide an extensive list of ways that names may be used in a CL expression or text, including
    - 6.1.1.9 Equation arguments,
    - 6.1.1.9 Atomic sentence predicates and arguments
    - 6.1.1.11 Functional term operators and arguments

XLCX and GML

In addition to their use as terms, names are used in a few other places.

1. 6.1.1.1 Named Text - associates a name N with the text
2. 6.1.1.4 Module - the module name denotes the module body text, and its relational extension defines a local universe of discourse, while the exclusion list gives a list of names excluded from that local universe.
3. 6.1.1.5 Importations - requires the name of named text to be imported
4. 6.1.1.7 Quantifications - require a list of bound names, possibly constrained.

In CL dialects, there is no requirement for or against a syntactic distinction between free and bound names, or names that refer to entities outside of the universe of discourse (UD). The CL abstract syntax allows for the assignment of a fixed interpretation to some names, called interpreted names, such as the single-quoted strings and the unquoted numerals of CLIF described in the next section, but leaves it open for a dialect to specify which names have such fixed interpretations and how to distinguish them from interpretable names.

Names in CLIF

The CLIF syntax, defined in Annex A of the CL Standard (ISO/IEC 2007) has four syntactic categories for names:

1. enclosed names, delimited by double quotes, the most general form for interpretable names, e. g., the name **`"Pacific Ocean"`** with the corresponding body of water as referent. The name **`"Océano Pacífico"`** has the same referent.
2. non-enclosed names, a short-hand form for enclosed names that are unambiguously represented without the surrounding double quotes, e. g., the name **`France`** referring to the country as an administrative entity.
3. strings, in particular sequences of Unicode characters (restricted by the requirement to escape [`' \`] as [`\' \\`]), delimited by single quotes, with a fixed interpretation that they refer to the contained string, e. g., the name **`'Pacific Ocean'`** with the referent of the character string **`'Pacific Ocean'`**. The string **`'Océano Pacífico'`** has a different referent.

4. numerals, i. e., sequences of the digits 0-9 with no quotations, having a fixed interpretation as non-negative integers, e. g., the name `123` with referent the integer one-hundred twenty-three. The name `0123` has the same referent.

Names in IKL Syntax

The IKL language (Hayes and Menzel 2006?) is very similar to CLIF, and so has `'quoted strings'` for character sequences, unquoted numerals, `123`, for numbers and `"enclosed names"` for interpretable names. However IKL also contains some additional syntactic forms.

- Datatyped names are constructed as an application of a function, e. g.,
  ```
  (xsd:float '1.2e-4')
  ```
- Named proposition are constructed with a special "`that`" syntax, e. g.,
  ```
  (that (and (P x) (Q y)))
  ```
- Captured strings are used to create interpretable names from character sequences. One form of the syntax uses a nullary function,
  ```
  (= ('abc') "abc")
  ```
  another uses a unary function "`tnb`" (for 'thing named by'),
  ```
  (= (tnb 'abc') "abc")
  ```

Names in XCL1

The XCL1 syntax uses a simple `<var/>` element for interpretable names as bound variables and `<term/>` for free interpretable names; thus, it is a segregated dialect. For example, a universal quantification could be written as follows:

Code Block 3.1: An XCL1 universal quantification.

```
<forall xmlns="http://purl.org/xcl/1.0/">
  <var name="x"/>
  <atomic>
    <relation>
      <term name="contains"/>
    </relation>
    <term name="Region A"/>
    <var name="x"/>
  </atomic>
</forall>
```

stating "**`Region A`**" is a bounding spatial region containing everything that is open for discussion, i. e., that is in our universe of discourse. For comparison, the same statement in CLIF is:

Code Block 3.2: A CLIF universal quantification.

```
(forall ("x") (contains "Region A" "x"))
```

Names in XML

A number of different ways have been developed in XML for representing names, including URIs, IRIs, relative references, datatyped simple content, qualified names, and Compact URIs (CURIEs).

URIs, IRIs and Relative References

According to (Berners-Lee, Fielding and Masinter 2005), a URI reference is either a URI or a relative reference. One form of a relative reference, familiar to users of HTML, is the "same-document" reference, where the relative reference is intended to be merged with a base URI. Independent of the "scheme" (the letters of a URI before the first colon), URIs may be a locator, a name or both. Unicode-based IRIs (Duerst and Suignard 2005) and IRI references are an internationalized extension of URIs and URI references, resp., that are supported by most XML applications, including the XML Schema datatypes discussed in the next section. All absolute (non-relative) URIs and IRIs have fixed denotations, independent of the language used or the context.

Datatypes

Datatypes are defined as a mapping from a lexical space of character strings to a value space of concepts (W3C 2004c). The mapping may be many-to-one, and in such cases the lexical space may be partitioned into equivalence classes, where a class consists of all members of the lexical space that denote the same value. Often, a canonical representation will be defined along with the datatype, as a unique character sequence from each equivalence class that thus stands in a 1-1 relationship with the value associated with the class. For example, the number one may be represented by `1` and also by `01` in the XSD integer datatype, while `1` is the canonical representation.

The XSD datatypes may be identified as qualified names in the namespace "`http://www.w3.org/2001/XMLSchema`" or as IRIs. The prefix "`xs`" is often used for this namespace, but any prefix could be used, and in any case the prefix must be declared

in the XML document. The IRI for an XSD datatype is obtained by concatenating the namespace with the local name separated by the character '`#`'. For example, the IRI for the "`xs:decimal`" datatype is "`http://www.w3.org/2001/XMLSchema#decimal`".

XSD schemas allow the "`xsi:type`" attribute by default (W3C 2004b), which accepts value as qualified names.

In the Relax NG schema language, a datatype attribute must be explicitly introduced, as well as the link between the datatype name and the content to which the datatype restriction is to apply(Vlist 2003). In a Relax NG schema, the XSD built-in datatypes may be imported directly, and there is also Relax NG syntax to generate some derived datatypes from the XSD built-ins by restricting facets such as string length or bounds on numerical values.

For datatypes that cannot be directly derived in Relax NG, this schema language has the capability to import user-defined datatypes written in common programming languages, such as Java. The implementation of a datatype must contain a validation method to check input for membership in the lexical space, a constructor to generate an object representing the value, a method to compare two inputs to see if they represent the same value, a hash value generator, methods to check for ID constraints and context dependence, and an optional streaming validator (Harold 2004).

Qualified Names, IRIs and CURIEs

Qualified names are ubiquitous in XML documents, so it seems natural that we should design XCLX to handle qualified names like any other name in the vocabulary. Although a qualified name is represented as "`ns:L`" where '`ns`' is the prefix and '`L`' is the local name, its use in an XML document is not valid unless the prefix is associated with a namespace identified by an IRI. The namespace declaration often is placed in the root element so that it may be applied in all contained elements. As an example, all XCLX documents must declare a prefix for the namespace `"http://www.example.org/xclx/1"`, as in Code Block 3.3.

Code Block 3.3: XCLX namespace declaration

```
<xclx:XCLX xmlns:xclx = "http://www.example.org/xclx/1">...<xclx:XCLX>
```

A default namespace may be declared as follows:

Code Block 3.4: XCLX namespace declaration as default

```
<XCLX xmlns = "http://www.example.org/xclx/1">...<XCLX>
```

We will use this convention throughout the following examples in order to save space. Please note that all the attributes in the XCLX syntax are either local (no namespace) or in a foreign namespace, so there is no ambiguity in presentation when using the XCLX namespace as default.

Namespace authorities will sometimes provide equivalent IRIs for qualified names, but there is no standard approach for combining a namespace and local name to obtain an IRI. The Clark notation (Clark 1999a), where the namespace is surrounded by curly brackets before the local name is appended, provides a means for serializing a qualified name. For example, the local name Atom in the RuleML namespace "**`http://www.ruleml.org/0.91/xsd`**" may be written "**`{http://www.ruleml.org/0.91/xsd}Atom`**". This is not a valid IRI, but it is a character sequence that provides an identifier across namespaces as a referent to the subclass of logical expressions that are applications of a predicate, which we will use when needed.

The more recently introduced CURIE datatypes (Birbeck and McCarron 2010) allows IRIs to be expressed compactly using prefixes which represent a character sequence that is prepended to the local name. Prefixes are declared using exactly the same notation as qualified names, and a CURIE with no prefix is prepended with the default namespace. The lexical space of CURIEs is a super-set of the lexical space of qualified names. However, the value space of this datatype is entirely different from the value space of qualified names, and is in fact identical to the value space of the datatype "**`xs:anyURI`**"[5].

5 In spite of the name, the datatype for IRIs is "**`xs:anyURI`**".

Names in RuleML

In the main branch of RuleML, names are segregated so that the names used for functions, relations and individuals belong to disjoint subsets of the vocabulary. There are also separate syntactic forms for literal constants, non-literal constants, skolem constants and variables.

Names in GML

According to the Reference Model (ISO 2005), the syntax categories are

- feature names,
- attribute names, or
- attribute values.

For example, in Code Block 1.4, "`#G3`" as a relative URI [11] is a feature name, "`nws:location`" as a qualified name [12] is an attribute name, and the XML contained within the element `<nws:location/>` on lines [13-16] is the attribute value. In this case the attribute value is complex content, but is not itself a feature, but rather is a geometry object.

GML implements this requirement of the Reference Model by denoting:

- feature names as descriptive names or identifiers,
- "attribute" names as the qualified names of the "property" elements,
- "attribute" values as denoted by the contents of the "property" elements;
    - if the attribute value is literal data, then it is denoted by the (simple) content of the property element;
    - if the attribute value is a feature, then it is denoted by the feature name or identifier associated with the (complex) content of the property element.

The GML in Code Block 1.4 expresses, among other things, the existence of an information object having a property "`gml:pos`" with attribute value "`29.5 -123.05`". Additional explicit information in the statement are the facts that this object belongs to a

sub-class of "**gml:Point**" that consists of 2D points associated with the spatial reference system "**urn:ogc:def:crs:epsg:4326**". Implicit in the statement is the information that such points are uniquely specified by two coordinates, the units of the coordinates are degrees, the first coordinate (latitude) is restricted to [-90, 90], different coordinates correspond to different points, except at the poles and dateline, etc.

GML requires all object elements to have a **@gml:id** attribute, providing identifiers satisfying a Unique Name assumption, i. e., two names denote the same thing if and only if the names are identical.

Other requirements are stated in (OGC 2007b), including:

- "A feature is encoded as an XML element with the name of the feature type. Other identifiable objects are encoded as XML elements with the name of the object type." For example, "**gml:Point**" is the class of the object identified by the relative IRI "**#G4**" in Code Block 1.4
- Properties are specified by the child elements of the objects. For example, "**gml:pos**", or rather its equivalent as a (namespace, local name) pair, identifies a relation.
- Property values are given by the contents of the property element, which may be encoded inline or referenced as a Simple Xlink.
- "the content model for all identifiable objects shall derive from **gml:AbstractGMLType**, and for all features from **gml:AbstractFeatureType**."

  In practice, this means that the attributes and sub-elements that are allowed in the abstract types must also be allowed on elements representing identifiable objects and features.

3.2.a XCLX Name Specifications

The Core syntax of XCLX will implement names as required for full CL conformance, using syntax somewhat similar to RuleML, but not segregated either in regard to names used as functions or relations, or with respect to bound or free variables. An XCLX extension will implement a unified name framework, with every name having two (for

interpretable names) or three (for interpreted names) parts;

1. the name of the local interpretation (optional);
2. the datatype;
3. the lexical part.

Every name in the syntax may be represented using a single element having a modifying attribute to specify the local interpretation, and a child element having a modifying attribute to specify the datatype to be applied to the lexical part of the name, the last appearing as the content of the child element. Universal names will be identified for the local vocabularies of literals and well-formed formulas.

This three-part naming framework will contribute to the application of contextual information, such as time of validity, to GML features and XCLX texts.

Section 3.3 Terms

In the CL dialects, compound terms, or functions, may be used in any position where a term is used in the syntax, including the operators of atomic sentences and the operators of compound terms themselves. On the surface this seems to move into the territory of higher-order logic, but as implemented in CL, it can be shown that it is actually still within the bounds of first-order logic (ISO/IEC 2007).

In FOL RuleML, functions are not allowed to be used as operators in atomic sentences or functions.

Terms in GML are names of features or properties, or literal data (ISO 2005). There is no explicit provision for terms as functional expressions.

3.3.a XCLX Term Specifications

The Core syntax of XCLX will implement terms as names and functions as required for full CL conformance, using syntax similar to RuleML. The functional syntax will allow the expression in XCLX of transformations of spatial reference systems, the construction of complex geometry objects from simpler objects, and the application of spatially-aware computational models to GML structured data.

Section 3.4 Term Sequences

The most interesting feature related to term sequences among the CL dialects is the notion of sequence markers. As a free variable, this is a convenient notation for a particular finite sequence of terms. As a bound variable, the sequence marker stands for a finite sequence of terms ***of arbitrary length***. This construct is particularly useful in structural axioms of polyadic (arbitrary number of arguments) syntactic forms.

RuleML has a similar form in the rest variable, but the syntax constrains its use to the last position of a term sequence, causing it to be less powerful than the CL version. There is no sequence marker syntax in GML.

3.4.a XCLX Term Sequence and Sequence Marker Specifications

The Core syntax of XCLX will implement the term sequence and sequence markers as required by full CL conformance, using syntax similar to RuleML for term sequences and introducing a novel syntax for sequence markers. An XCLX extension will implement a unified functional term framework, with a single element having one or more modifying attributes to specify the kind of function. Another XCLX extension will implement unified **`<term/>`**/**`<terms/>`** elements, taking one /(zero to many) XCLX terms as sub-elements, and having one or more optional modifying attributes specifying the parent element and any distinguishing characteristics. This will implement the fully-striped syntax of the unified atomic sentence, equation and function frameworks, described below in Section 3.5-Section 3.8. It will also allow embedding of text written as foreign dialects, such as CLIF, as terms and the specification of a local vocabulary independently on individual terms of atomic sentences and functions.

Section 3.5 Equations

The CL semantics produces a subtle but powerful effect on equations because every name carries a triple denotation, as individual referent, function and relation, and the latter two are specified by a mapping from the individual referent to the function or relation, rather than from the name. Therefore, if we equate the individual referents of a pair of names, we also equate the functional mapping and relational extension of that name as well.

IKL contains the same strong equality of CL, but also contains weaker forms, where the

equivalence of the relational extension or functional mapping associated with two names may be asserted, without asserting the equality of their individual referents.

RuleML equations are not as powerful as CL equations because, as a segregated dialect, names don't carry the triple denotation, as individual, function and relation, that is implied by the CL semantics.

There is no explicit equation syntax in GML.

3.5.a XCLX Equation Specifications

The Core syntax of XCLX will implement equations as required by CL conformance, using syntax similar to RuleML. An XCLX extension will implement a unified equation framework, with a single element having one or more modifying attribute to specify the kind of equation, allowing implementation of weaker kinds of equations, e. g., equivalence of relational extension. This syntax will allow the expression of more complex relationships among GML classes.

Section 3.6 Atomic Sentences

Non-fuzzy positional[6] atomic sentences in RuleML have a similar syntax and identical semantics as those of the CL dialects. In contrast, a GML "object" in an instance of an application schema is equivalent to three atomic sentences:

1. The thing denoted by the feature identifier is a member of the class denoted by the qualified name of the outer feature element.
2. The pair (feature identifier, attribute value) is a member of the extension of the relation denoted by the qualified name of the property element.
3. The attribute value is a member of the class denoted by the corresponding type of the attribute content model, as specified in the application schema.

The GML semantics is similar to what obtains in Description Logics such as OWL, where the domain and range specifications of a relation result in assertions on the subject and object of a property instance, rather than a restriction on the application of the

6 Both CL and RuleML also have slotted forms of atomic sentences, and RuleML also has fuzzy atomic sentences.

property (W3C 2004a).

3.6.a XCLX Atomic Sentence Specifications

The Core syntax of XCLX will implement the atomic sentence as required by CL conformance, using syntax similar to RuleML. This syntax will allow expression of the meaning of GML instances in terms of atomic sentences.

Section 3.7 Boolean Sentences

Boolean sentences have similar shortcut syntax and identical semantics in CLIF, XCL1, IKL, ICL, and RuleML. In additional, XCL1 has a unified framework for Boolean sentences based on the element **`<boolean/>`** with modifying attribute **`@syntaxType`**. There is no explicit syntax for Boolean sentences in GML.

3.7.a XCLX Boolean Sentence Specifications

The Core syntax of XCLX will implement shortcuts for the Boolean connectives required by CL conformance, using syntax similar to RuleML. An XCLX extension will implement a unified Boolean framework using syntax similar to XCL1, with a single element having one or more modifying attribute to specify the kind of connective, allowing implementation of, e. g., exclusive disjunction. This syntax will allow compounding of GML features, such as the use of a GML feature as a premise or conclusion in an implication.

Section 3.8 Quantification

The basic quantifications, universal and existential, have similar syntax for shortcuts and identical semantics in CLIF, XCL1, IKL, ICL, and RuleML. XCL1 also has a generic **`<quantified/>`** framework. IKL also has syntax for numerical quantification (there exists exactly two things such that ...).

Quantification may be expressed indirectly in GML through subclass relationships, which are a particular form of universal quantification. Subclass relationships are expressed in application schema through type inheritance, where naming conventions (**`gml:Point`**, **`gml:PointType`**) are used to relate type inheritance statements to the corresponding

subclass relationship. However in XSD, multiple inheritance of types is not allowed, so the class relationship structure is limited to a tree form.

Another type of quantification is expressed in application schema by annotation with **`<gml:reversePropertyName/>`**, which relates two properties so that if one is true of a pair (x, y) the other is true of (y, x).

3.8.a XCLX Quantification Specifications

The Core syntax of XCLX will implement the quantifiers required by CL conformance, using syntax similar to RuleML. An XCLX extension will implement a unified quantification framework similar to XCL1, with a single element having one or more modifying attribute to specify the kind of quantification, allowing implementation of, e. g., numerical quantification as in IKL. This syntax will allow quantification over parts of GML features, including feature names, attribute names and values.

Section 3.9 Context

Although "context" is recognized as an important topic in KR (McCarthy and Hayes 1969), psychology and philosophy, e. g. (Korta and Perry 2011), it is apropos that there is no single definition of the term, so that its meaning must be inferred from its context. McCarthy doesn't provide a definition at all. For those authors that do, the definitions of context may be divided into two senses: linguistic context and physical context.

Narrowly considered, linguistic context is the set of words nearby or surrounding a word or phrase that influence its meaning. The physical sense of context focuses on factors external to the verbalization. Certain parameters of context such as the physical location of a mobile device or characteristics of its user (Keßler, Raubal and Wosniok 2009) have been used in geographic information retrieval. A broader sense of context, presented in (Guha and McCarthy 2003), considers four facets: projection, approximation, ambiguity and mental state.

Contextual KR is found at the confluence of these two senses of context, in that a formal notation is employed, usually surrounding an expression, to denote the factors external to the expression which affects its meaning.

One of the properties of contextual knowledge is the possible limitation of the inference rules over which the context will "distribute". The presence of a distributive property is called "transparency" in the contextual logic literature, its absence is "opacity" (IKRIS 2007b). Contexts are often opaque to disjunction (inclusive or) and negation. Modifying an example given in (IKRIS 2007b), suppose the context is "is true on Tuesday", and the proposition is "the air temperature is above 30 ºC". The sentence "On Tuesday, at any particular time, the air temperature is either above 30 ºC or not above 30 ºC" is clearly tautologically true. However the sentence "Either the air temperature is above 30 ºC at any particular time on Tuesday, or the air temperature is not above 30 ºC at any particular time on Tuesday" is not necessarily true - it is quite possible for the temperature to be above 30 ºC for parts of the day and below for others.

Similar examples arises in several of the contextual facets relevant to geographical data, particularly in regard to aggregated data. The well-known "ecological fallacy" (Robinson 1950), whereby correlations of the statistical properties of populations are erroneously extrapolated to individuals, is another example of contextual opacity.

At least three different formal approaches to context have emerged in KR, with superficially similar syntax but distinctly different semantics, including:

- contexts as modal operators, applied to sentences;
- contexts as predicates, applied to names of propositions;
- context as a special syntactic category.

CL Modules

The abstract CL semantics contains a limited notion of context as the "**`module`**" construct. This falls into the third type of context approach, as "**`module`**" is an extra-logical syntactic category.

The relational extension of the name of a module N is defined to be exactly the set of finite sequences of discourse names for the module. In essence, a local interpretation is created that is identical to the interpretation applied to the text as a whole, except for the assignment of certain names as non-discourse names. The CLIF semantics extends the abstract CL semantics rather than implementing it directly. Modules are implemented as

a conjunction of statements asserting that excluded names do not belong to the extension of N, as well as asserting a modified version of the module body, with bindings constrained by the relation N. While not equivalent, the CLIF module semantics is a necessary condition of the abstract CL module semantics, and that is all that is required of a CL extension.

Context in IKL

Modules are implemented in IKL as they are in CLIF, but IKL also contains the "**`that`**" construct, which ensures that all syntactically valid sentences, also call "well-formed formulas" (wffs), have names in the vocabulary. These names can be used as arguments of atomic sentences, and context is implemented in CLIF as the second type of context approach, as a predicate applied to the name of a proposition. This syntax allows contexts to be nested.

Code Block 3.5: A nested context sentence in IKL.

```
( (believes Jane) (that ( (said Ali) (that (likes Jose Olga)) )) )
```

The contextual transparency to logical operators can be defined axiomatically for contextual sentences. For example, it may be asserted that all contexts are "A-clear", meaning the context relation distributes over conjunction, as follows[7]:

Code Block 3.6: An IKL axiomatic declaration that all contexts are A-clear.

```
(forall ((c Context) (S1 wff) (S2 wff))
  (iff
    (and (c S1) (c S2))
    (c
      (that
        (and (S1) (S2))
) ) ) )
```

7 The axiom shown here only address the case of binary conjunction; the axiom for polyadic conjunction is more complex.

Context in ICL

ICL implements context as a special syntactic form "**`(ist c S)`**" which takes a sentence "**`S`**" and a context name "**`c`**" as arguments. The expressive power induced by ICL's "**`ist`**" is less than that of IKL - using the "**`that`**" syntax, IKL can express every ICL statement using "**`ist`**", but not the reverse, because ICL does not allow quantification over propositions.

ICL also introduces a reserved word "**`specs`**" for the notion that one context "specializes" another. This can also be implemented in IKL without special syntactic structures, through the following axiom, where "**`specs`**" is simply a name used as a binary relation.

Code Block 3.7: An IKL axiomatic declaration that a context c2 specializes context c1.

```
(forall ((c1 Context) (c2 Context))
  (iff (
    (specs c2 c1)
    (forall ((S wff))
      (if (c1 S) (c2 S))
) ) )
```

Finally ICL introduces "pragmas" which specifies the inference rules that apply for a particular text. For example, the statement "**`(ist-AON T)`**" would mean that the (implicit) context of the text T is distributive over the Boolean connectives and (A), or (O) and negation (N). Such statements are beyond the expressive power of IKL. Unfortunately, ICL does not appear to be under active development, with the ICL User's Guide published online as an unfinished draft (IKRIS 2007b).

Context in RuleML

Although not yet implemented, RuleML design documents (Boley, Paschke and Shafiq 2010) propose syntax for context following the first type of approach, as modal operators applied to sentences.

Context in GML

Metadata provides much of the context for GML structured data, and GML provides syntax to link from any element to metadata that might be relevant to the interpretation of

the data contained in that element. Additional aspects of context are contained in the schema documents and meta-models. An identifier to encapsulate the full context of a GML dataset could be, e. g., the complete IRI, including query string, of the Web Feature Service (WFS) where the dataset is published, appended with a suffix with a time-stamp for the time of access.

3.9.a XCLX Context Specifications

The Core syntax of XCLX will implement the CL module construct as syntactic sugar following the approach of CLIF. An XCLX extension will implement a syntax equivalent to the IKL "`that`", allowing XCLX to express any contextual knowledge that can be expressed in IKL, with an IRI for a context identifier. When combined with the capability for embedding GML and the meta-language, this will allow formal expression of contextual knowledge about GML datasets, including the time interval of validity, provenance and quality of data.

Section 3.10 Phrases and Texts

Extra-logical components of KR languages allow the logical components to be assembled into collections, organized hierarchically, commented, named, and imported into other texts. Structured data formats may have such capabilities as well.

Phrases and Texts in the CL Dialects

The principle functions of the "extra-logical"-level elements in the CL dialects is to associate names with texts so they can be imported elsewhere. Comments may be applied to texts without affecting the semantics. CLIF, IKL, and ICL adopt a similar strategy. However, there are problems, both typographical and conceptual, near the starting point of the CLIF grammar that make the productions ambiguous and awkward. The IKL starting productions are cleaner, although they also contain an error in the omission of the module construct from the phrase choice pattern. The ICL grammar has the oversight of omitting the phrase choice from their starting pattern, causing all texts to be named texts and hence never asserting anything other than the names of their texts.

Phrases and Texts in RuleML

The **<Rulebase/>** element is the mechanism for forming a text of several sentences. A RuleML rulebase is sensitive to order and duplication. The semantics of RuleML is not clear as to the role of the **<oid/>** element that is allowed in the content model of a **<Rulebase/>**. While it could potentially play the same role as the name of a named CL text, the assertion (with the **<Asserts/>** element) of a rulebase in RuleML does more than associate the identifier with the rulebase, but also asserts the truth of all of the member sentences.

Phrases and Texts in GML

The only means available in GML itself for describing multiple features is through a feature collection, which is itself a feature. An application schema has the additional capability to introduce a type corresponding to a "collection" of features that is not itself a feature, but instead is derived from the more general type "**AbstractObjectType**". This construct is the counterpart of texts in KR languages. An identifier may be supplied for the collection, enabling reification of datasets for the purpose of expressing metadata such as authority.

Schema languages, such as XSD or Relax NG, that may be used to build application schemas have the capability of including other schemas. This is a counterpart of the importation phrase in KR languages, but is considerably weaker in that an instance document cannot import external data.

Annotations in GML are the counterpart of comments in KR languages.

3.10.a XCLX Phrase and Text Specifications

The Core syntax of XCLX will satisfy the requirements of the CL abstract specifications through the implementation of the collection of phrases into a text with a document root element and a unified text element that may be empty, and accepts an optional name, for named texts, and an optional comment. The document root may contain zero to many text elements, and the text elements may not be nested. Additional elements will be implemented in the Core syntax for commented terms, sentences and texts, which may be nested to arbitrary depth. The Core syntax will also implement an importation element,

which may be annotated with processing information having no semantic content. At this time, XCLX extensions related to phrases or texts are not envisioned, but are not forbidden.

Section 3.11 Meta-Language

Meta-language is a language that may be used to express things about another language. Grammars and schema languages are external languages that fall into this category, and are used to specify the syntax of the KR languages we consider here, as discussed in Section 2.1. In this subsection, we compare the self-referential capability of these KR languages - their ability to describe and extend themselves.

The CL Standard (ISO/IEC 2007) does not include a meta-language capability, although it does not preclude such a capability in extensions.

Meta-Language in IKL

IKL, while for the most part similar to CLIF, has two novel features that enhance its self-extensibility, so that it has a meta-language, albeit with limited capabilities.

The syntactic form "**that**", described in Section 3.2, is used to create propositional names. Because of this property of IKL, we may axiomatically define semantics for extensions of the language. For example, Code Block 3.8 defines semantics for a new Boolean operator "**xor**" for exclusive disjunction[8]

Code Block 3.8: An IKL axiomatic declaration of a Boolean for exclusive disjunction.

```
(forall ((S wff) (T wff)) (iff
  (xor (S) (T))
  (or (and (S) not((T))) (and not((S)) (T)))
))
```

Although Code Block 3.8 does not explicitly use the "**that**" syntax, it implicitly uses the fact that the vocabulary contains names for all syntactically valid propositions (wffs).

Another kind of self-extensibility in IKL uses variables denoting character sequences

8 For illustrative purpose, we limit ourselves to the case of two arguments.

together with the captured string syntax, described in Section 3.2, to axiomatically define various naming conventions.

Meta-Language in RuleML

While RuleML doesn't have any explicit meta-language capabilities, it does have the **<Entails/>** meta-logic element. In the convention of traditional logic, there is an implicit universal quantification over all interpretations, and we may assume this as the semantics of this syntactic form[9]. For example, the following code asserts that equality is reflexive (x = x) for all x, in all interpretations:

Code Block 3.9: An entailment in RuleML.

```
<ruleml:Entails xmlns:ruleml="http://www.ruleml.org/0.91/xsd">
  <ruleml:Rulebase/>
  <ruleml:Rulebase> <ruleml:Forall>
    <ruleml:declare> <ruleml:Var>x</ruleml:Var> </ruleml:declare>
    <ruleml:Equal>
      <ruleml:Var>x</ruleml:Var> <ruleml:Var>x</ruleml:Var>
    </ruleml:Equal>
  </ruleml:Forall> </ruleml:Rulebase>
</ruleml:Entails>
```

However, there is no provision in RuleML for transforming character sequences into RuleML sentences as is required to achieve full self-extensibility of the language.

Meta-Language in GML

According to (OGC 2009), the meta-meta-model and meta-model levels of the conceptual model approach provide the syntax and semantics of the languages used to described GML application schemas, which in turn provide the syntax and (some of) the semantics of the language for instance data. The GML meta-meta-model is informal, but the meta-model has been formalized, using the Unified Modeling Language (UML). Thus, the meta-models may be considered meta-language documents for application schemas, while application schemas may be considered meta-language documents for instance data.

9 There are actually two kinds of entailment: proof-theoretic entailment and model-theoretic entailment, but these are equivalent within the semantics of RuleML.

3.11.a XCLX Meta-Language Specifications

The Core syntax of XCLX will not include meta-language capability, as this is not part of the CL abstract specifications. In the XCLX extensions, an **`<Entails/>`** element will be introduced, similar to the RuleML syntax. For full meta-language capability, there is no corresponding syntax in any of the languages considered. Therefore a novel syntax will be introduced, modeled after XSLT (Clark 1999b).

Section 3.12 Summary

Of the languages reviewed, the most significant influence on XCLX is the CL abstract syntax and semantics, because of the necessity of conformance with these in order for XCLX to qualify as a CL dialect. Extensions of XCLX that go beyond CL will draw their semantics primarily from IKL, and to a lesser extent from ICL. A design decision has been made that naming conventions and style of XCLX are drawn primarily from RuleML, that being the most widely-used XML-based KR language with first-order expressive power, and in keeping with the long-term objective of imbuing a subset of RuleML with Common Logic Semantics.

The XCLX schema will be made open to sentences and terms consisting of elements from foreign namespaces so that structured data such as GML may be embedded. To ensure that only syntactically-valid GML is embedded, validation of an XCLX document with embedded GML must include namespace-based validation against the appropriate schema(s), including the GML schema itself and zero to many application schemas.

Axioms that express the semantics contained in the GML meta-meta- and meta-models, as well as the relevant application schemas should be imported whenever GML is embedded in XCLX, so that this extra information is visible to inference engines and queries.

Names in the namespaces of GML and application schemas may be use directly in XCLX texts as qualified names. Contextual information may be necessary to clarify the sense in which the names are being used, e. g., when relevant semantic information is contained in external metadata.

Chapter 4 XCLX-ab

The XCLX language may be constructed incrementally. In Section 4.1, we define a limited syntax of basic elements that is a CL extension equivalent to CLIF, and contains a subset which is a fully conformant CL dialect. The basic elements form the Core syntax of the XCLX-ab (for abbreviated) flavor, the part of the XCLX language that is most compact, using short element names and minimizing the use of child elements and attributes while retaining insensitivity to order. In Section 4.2, we define the semantics of these elements. In Chapter 6 we describe some extensions to XCLX-ab. Further extensions of the XCLX language are discussed in Chapter 7 - 10.

Section 4.1 Core XCLX-ab Syntax

The XCLX basic syntax is defined by a Relax NG schema (ISO/IEC 2008). Validity was established using the oXygen XML editing software package (Syncro Soft Ltd. 2011), which provides a GUI interface to the Jing-Trang software (Clark 2009).

The schema shown in Code Block 4.1 below is derived from the modularized version of the schema in the attached electronic files, directories "**`drivers`**", "**`modules`**" and "**`suites`**", that was developed following the schema design pattern presented in (Athan and Boley in press). The Jing software package (Clark 2009) was applied to simplify the modular version into a monolithic, compact form, which was then manually modified for enhanced readability. The modular version of the schema may be customized by removing unneeded modules or extended by adding new ones, and is the version used in practice for validation.

Many of the symbols used in the Relax NG compact syntax having meanings similar to their usage in regular expressions (and thus the 'Re' in Relax NG). The symbol '|' indicates a choice (a|b matches a or b), parentheses are used for grouping, and the number of repetitions of a group is indicated by ?, * or + (zero or one, zero to many, one to many, resp.).

There are a few other symbols which either have a different meaning than in regular expressions, or do not have a counterpart in regular expression syntax. '&' indicates interleaving of sequences while retaining the relative order of each ((a & (b c)) matches

(a b c), (b a c), or (b c a)). The symbol ',' indicates a sequence of named patterns. Reserved words include 'start' (unsurprisingly, the start pattern) and 'element' followed by the element's name or wildcard[10] * and then the element's content model in curly brackets. Attributes are defined similarly. Datatypes such as "**`xsd:anyURI`**" produce restrictions to simple content of elements or attributes. The most complex production in this syntax is the datatype for compact URIs, also known as CURIEs (Birbeck and McCarron 2010) on lines [8-9], which uses the limited regular expression syntax of Relax NG to create a restriction of the XSD schema "**`xs:string`**" datatype (W3C 2004c).

10 '*' has 2 contextual meanings.

Code Block 4.1: Syntax for Core XCLX-ab as a simplified, monolithic Relax NG schema.

```
1  default namespace = "http://www.example.org/xclx/1"
2  start = XCLX
3  XCLX = element XCLX { (CLText | phrase)* }
4  CLText = element CLText { textname? & comment? & phrase* }
5  textname =
6    attribute name { xsd:anyURI }
7    | attribute cname {
8        xsd:string { minLength = "1"
9                     pattern = "(([\i-[:]][\c-[:]]*)?:)?.+" } }
10 comment = element comment { text & anyElement* }
11 anyElement = element * {
12   attribute * { text }*, (text & anyElement*) }
13 phrase = Importation | CommentedText | sentence
14 Importation =  element Importation { textname &
15     attribute textLocation { xsd:anyURI }? & fallback? }
16 fallback = element fallback {
17   attribute textLocation { xsd:anyURI }? & fallback? }
18 CommentedText = element CommentedText { comment & phrase* }
19 sentence = CommentedSentence|
20   And | Or | Implies | Equiv | Not | Forall | Exists | Equal | Atom
21 CommentedSentence = element CommentedSentence {
22   comment & sentence }
23 And = element And { sentence* }
24 Or = element Or { sentence* }
25 Implies = element Implies { sentence, sentence }
26 Equiv = element Equiv { sentence, sentence }
27 Not = element Not { sentence }
28 Forall = element Forall { binds+ & sentence }
29 Exists = element Exists { binds+ & sentence }
30 binds = element binds { bname+ }
31 bname = Var
32 Equal = element Equal { term, term }
33 Atom = element Atom { op & termseq }
34 op = element op { term }
35 termseq = (term | Seq)*
36 term = Expr | CommentedTerm| name
37 Expr = element Expr { op & termseq }
38 CommentedTerm = element CommentedTerm { comment & term }
39 name = Var | Data | Num
40 Var = element Var { xsd:token }
41 Data = element Data { xsd:string }
42 Num = element Num { xsd:nonNegativeInteger }
43 Seq = element Seq { xsd:token }
```

We now discuss the syntax presented in Code Block 4.1. Line numbers are referenced as a number in brackets, e. g. [1]. XML elements are denoted as empty elements, e. g. `<XCLX/>` and attributes with a leading "@", e. g. `@name`.

Throughout this dissertation, we use a placeholder namespace for XCLX-ab [1] of "`http://www.example.org/xclx/1`". There is a long-term aim, jointly held by the CL

and RuleML communities, to have a joint XML-based CL dialect, XCL version 2, that would use the CL semantics and the RuleML syntax, would replace

XCL version 1, would be released in the next revision of the CL standard, and would

also serve as a new subfamily, CL RuleML, of RuleML (Boley 2010). XCLX was developed as the next step from XCL version 1 to the planned XCL version 2 / CL RuleML with a focus on GIS applications. During the development of XCLX, substantial input was received from these two communities. However, there is still negotiation needed before a final version is released, including consideration of naming collisions with other extensions of RuleML. Therefore the final namespace for XCLX is not specified at this stage.

The syntax of the first few nodes of the document hierarchy, which we call the "extra-logical" level, are described in lines [2-18] of Code Block 4.1. In Core XCLX-ab, all documents start with the **<XCLX/>** element [2]. There may be several **<CLText/>** elements in the document [3], but this element cannot be nested [4]. **<CLText/>** may contain any number of **phrase** patterns, as well as an optional name and/or comment [4]. Alternately, **<CLText/>** can be skipped and the **phrase** pattern can appear as a direct child of **<XCLX/>** [3]. A text may be named as an IRI [6] or a CURIE [7-9]. Comments may contain mixed text and arbitrary well-formed XML [10-12].

The choice pattern **phrase** matches either an **<Importation/>**, a **<CommentedText/>** or a **sentence** pattern [13]. A **<CommentedText/>** is similar to an unnamed **<CLText/>**, but can be nested [18]. An **<Importation/>** must reference a named text, which may be in any dialect (XCLX, CLIF, ...) [14]. Optional components for **<Importation/>** are a **@textLocation** attribute and a **<fallback/>** element [15], which can be nested [16-17]. These components provide processing instructions to assist an application in locating the text to be imported on the network, and should be implemented like the corresponding components in XML Inclusions (XInclude) (Marsh, Orchard and Veillard 2006).

An example XCLX-ab (instance) document demonstrating the "extra-logical" level is shown in Code Block 4.2.

Code Block 4.2: Sample XCLX-ab demonstrating extra-logical level.

```
1  <XCLX xmlns="http://www.example.org/xclx/1">
2    <CLText cname="/ex/ab#extra">
3      <comment>This is a named text that may be asserted by
4        importation.</comment>
5      <CommentedText>
6        <comment>This is a commented text, which just attaches this
7          comment to the following text.</comment>
8        <And/>
9        <!-- This is an informal comment, which may not be
10            preserved in translation to other dialects. -->
11     </CommentedText>
12   </CLText>
13   <Importation cname="/ex/ab#extra" textLocation=".">
14     <fallback textLocation="http://www.example.org/extra.xclx"/>
15   </Importation>
16 </XCLX>
```

The "logical" levels of the document hierarchy contain the majority of the semantic content of the document, and these levels all begin with something matching the **sentence** choice pattern that is a child of **<XCLX/>**, **<CLText>** or **<CommentedText/>**. The **sentence** choice pattern matches several sentence types (see lines [19-20] of Code Block 4.1), whose syntax is elaborated in lines [21-34]. Most of these sentence types are compound sentences, containing other sentences. The "logical" level terminates in the following ways:

- An empty **<And/>** or **<Or/>** sentence, which correspond to the propositional constants True and False, resp.;
- An atomic sentence, which is the application of a predicate (**<op/>**) to a term sequence (**termseq**) [33-35];
- An equation, applied to two **terms** [32].

An example XCLX document illustrating the "logical" level is shown in Code Block 4.3.

In most cases, the logical level will terminate with a number of **term** choice patterns, which may occur as predicates, arguments of atomic sentences or the sides of an equation. Alternately, a term sequence may be indicated by a sequence marker (**<Seq/>**) [35, 43].

Code Block 4.3: Sample XCLX-ab demonstrating some of the logical level.

```
1  <Implies>
2    <Equal> <Var>P</Var> <Var>Q</Var> </Equal>
3    <Forall>
4      <binds> <Var>x</Var>
5      </binds>
6      <Equiv>
7        <Atom> <op> <Var>P</Var> </op>
8          <Var>x</Var> </Atom>
9        <Atom> <op> <Var>Q</Var> </op>
10         <Var>x</Var> </Atom>
11     </Equiv>
12   </Forall>
13 </Implies>
```

Terms may be a compound term, a commented term or a name (see line [36] of Code Block 4.1). Compound terms are constructed with the <**Expr/>** syntax [37], while the commented term syntax is given in line [38]. The name choice pattern includes **<Var/>**, **<Data/>** and **<Num/>**[39-42].

An example XCLX document illustrating the "term" level is shown in Code Block 4.4.

Code Block 4.4: Sample XCLX-ab demonstrating term level.

```
1  <Equal>
2    <Expr> <op> <Var>xs:decimal</Var> </op>
3      <Data>0123</Data> </Expr>
4    <Num>123</Num>
5  </Equal>
```

The syntax for the logical and term levels of XLCX-ab is mostly equivalent to RuleML (aside from the namespace). The primary departures from RuleML are:

- use of **<binds/>** instead of **<declare/>** and **<Not/>** instead of **<Neg/>**
- multiple children allowed within the **<binds/>** element, as opposed to a single child within **<declare/>**

Section 4.2 Core XCLX-ab Semantics

In this subsection, we define the semantics of the Core XCLX-ab by translation into

CLIF (ISO/IEC 2007). Variables are denoted by bold text in Arial font (**n**). Independently within each row, the variables may be replaced by any character sequence that results in both valid XML in the left column and valid CLIF in the right column, subject to the caveats expressed in the indicated footnotes. The label of the row for the corresponding CL Abstract syntax is given in the last column of Table 1. The symbol "**&**" is used with the same meaning as in Schema 3.1, to indicate interleaving.

Table 1: Syntactic forms of Core XCLX-ab translated to their CLIF counterparts

	XCLX Expression **E**	CLIF Expression T[**E**]	Note	CL
1	<Data>**n**</Data>	'**s**'	a	E1, E3
2	<Num>**j**</Num>	**j**	b	E1, E3
3	<Var>**m**</Var>	"**t**"	c	E1, E3
4	<Seq>**m**</Seq>	"...**t**"	c	E2, E4
5	<Expr><op>**F**</op> & **T**$_1$ ... **T**$_n$ </Expr>	(T[**F**] T[**T**$_1$]...T[**T**$_n$])	d	E5
6	<Atom><op>**F**</op> & **T**$_1$ ... **T**$_n$ </Atom>	(T[**P**] T[**T**$_1$]...T[**T**$_n$])	e	E7
7	<Equal> **T**$_1$ **T**$_2$ </Equal>	(= T[**T**$_1$]T[**T**$_2$])		E6
8	<And> **S**$_1$ ... **S**$_n$ </And>	(and T[**S**$_1$]...T[**S**$_n$])		E9
9	<Or> **S**$_1$ ... **S**$_n$ </Or>	(or T[**S**$_1$]...T[**S**$_n$])		E10
10	<Implies> **S**$_1$ **S**$_2$ </Implies>	(if T[**S**$_1$] T[**S**$_2$])		E11
11	<Equiv> **S**$_1$ **S**$_2$ </Equiv>	(iff T[**S**$_1$] T[**S**$_2$])		E12

a The value of **n** is restricted to the lexical space of the datatype xs:string. The character sequence **s** is obtained from **n** by escaping the characters [\ '] with [\\ \'], respectively, and translating the named character entity references [< >] as [\u003C \u003E], respectively. The latter are the Unicode representations of [< >]. In CLIF, the single-quotes wrapping the character sequence indicate it is a literal name denoting the character sequence **n**.

b The value of **j** is restricted to numerals, i.e., character sequences containing only the digits [0-9].

c The value of **m** is restricted to the lexical space of the datatype xs:token. The character sequence **t** is obtained from **m** by removing leading and trailing whitespaces, replacing all whitespace sequences by a single space, and escaping the characters [\ "] with [\\ \"], respectively, and the named character entity references [< >] with [\u003C \u003E], respectively. In CLIF, the double-quotes wrapping the character sequence indicate it is a non-literal name denoting something in the universe of reference.

d The inverse translation from the CLIF (f x1 ... xn) into an **<Expr/>** element is applied to CLIF compound terms.

e The inverse translation from the CLIF (P x1 ... xn) into an **<Atom/>** element is applied to CLIF atomic sentences.

	XCLX Expression **E**	CLIF Expression T[**E**]	Note	CL
12	<Not> **S** </Not>	(not T[**S**])		E8
13	<**Q**> <binds> **T**$_1$ ... **T**$_n$ </binds> & <binds> **T**$_{n+1}$ ... **T**$_m$ </binds> & **S** </**Q**>	T[<**Q**> <binds> **T**$_1$ ...**T**$_m$ </binds> & **S** </Q>]		
14	<Forall> <binds> **T**$_1$ ... **T**$_n$ </binds> & **S** </Forall>	(forall (T[**T**$_1$]...T[**T**$_n$]) T[**S**])		E13
15	<Exists> <binds> **T**$_1$ ... **T**$_n$ </binds> & **S** </Exists>	(exists (T[**T**$_1$]...T[**T**$_n$]) T[**S**])		E14
16	<CommentedTerm> <comment>**n**</comment> & **T** </CommentedTerm>	(cl-comment '**s'** T[**T**])	a	
17	<CommentedSentence> <comment>**n**</comment> & **S** </CommentedSentence>	(cl-comment '**s'** T[**S**])	a	
18	<CommentedText> <comment>**n**</comment> & **P**$_1$ ... **P**$_n$ </CommentedText>	(cl-comment '**s'** T[**P**$_1$]...T[**P**$_n$])	a	
19	<**a** cname="**b:c**" xmlns:**b**="**n**">**S**</**a**>	T[<**a** name="**nc**">**S**</**a**>]		
20	<Importation name="**u**">**S**</Importation>	(cl-imports "**u**")	f	E17
21	<CLText **q**> <comment>**n**</comment> & **P**$_1$ ...**P**$_n$ </CLText>	T[<CLText **q**> <CommentedText> <comment>**n**</comment> & **P**$_1$ . . . **P**$_n$ </CommentedText> </CLText>]		
22	<CLText> **P**$_1$ ... **P**$_n$ </CLText>	T[**P**$_1$]...T[**P**$_n$]		E19
23	<CLText name="**u**"> **P**$_1$ ... **P**$_n$ </CLText>	(cl-text "**u**" T[**P**$_1$] . . . T[**P**$_n$])	f	E20
24	<XCLX> **P**$_1$ ... **P**$_n$ </XCLX>	T[**P**$_1$] . . . T[**P**$_n$]		

Note that we have not included the CLIF "sweet" forms (Table A.2 of (ISO/IEC 2007)) in Table 1. There are several reasons for this:

f The value of **u** is restricted to the lexical space of the datatype "**xs:anyURI**".

- Role set syntax is omitted from this discussion due to space limitations.
- The syntax for "constrained" binding is defined below in Table 3.
- The validity of the guarded binding (ISO/IEC 2007) has been questioned in the defect report (International Organization for Standardization 2008).
- Modules (ISO/IEC 2007) will be discussed in Section 9.1.

Certain components of the syntax do not affect the semantics but do have an effect on processing a text. There are several such features in the importation syntax, which were inspired by similar features of the XInclude (Marsh, Orchard and Veillard 2006) syntax. The `@textLocation` attribute of an `<Importation/>` element provides a hint as to where to find a text with the specified name. If the text is not found at that URL, or the URL is not available, then a fallback element may be consulted to find a second URL to try. The fallback element may be nested to provide an arbitrary number of suggested URLs. More generally, processing instructions may be contained in comments, which are ignored from a semantic perspective.

Chapter 5 XCLX Extensions

There are many common uses of KR languages that are not satisfied by the core language presented in Chapter 4, and hence most applications of XCLX will make use of some extension of the language. There is a variety of ways in which the language may be extended. In each case it is necessary to extend both the syntax and the semantics.

Section 5.1 Methods for extending the syntax of XCLX

5.1.a Schema-based, same namespace

This method adds elements and/or attributes to the core XCLX namespace that are not present in the core schema. This method is appropriate only for "syntactic sugar", which provides alternate syntactic forms for core expressions and does not extend the semantics. Some extensions that meet this requirement are discussed in Chapter 6.

5.1.b Schema-based, new namespace

This method adds elements and/or attributes to the syntax that require semantics outside of the CL semantics. This method is well-suited for general purpose extensions, and some, such as the unifying **`<Name/>`**, **`<Boolean/>`** and **`<Quantification/>`** frameworks, are briefly describe in Chapter 8.

The semantics of these extensions may be defined by any of the methods listed in Section 5.2.

5.1.c Open modifying attributes

This method builds off of the unifying frameworks developed in Chapter 6 and Chapter 8. In those extensions, the modifying attributes (such as **`@datatype`** and **`@kind`**) are restricted to specific, enumerated values. These forms may be easily extended by allowing the modifying attributes to take on any IRI (or CURIE, as appropriate) values.

Once the syntax has been opened in this fashion, it becomes unavoidable that certain XCLX instances may be constructed that are syntactically valid but semantically undefined. In this case the axiomatic method of extending the semantics (Section 5.2.c) becomes vital, so that the semantics may be made available to applications. Applications

should be able to check whether the semantics of any syntactic forms used in an instance are undefined, and issue appropriate error or warning messages.

5.1.d Open sentence, term and name patterns

This method allows elements from foreign namespaces to be used in any position available to a particular syntactic category, and is implemented by Relax NG code such as the following, which references patterns defined in Code Block 4.1:

Code Block 5.1: XCLX schema expansion for open sentences and terms.

```
sentence |= anyElement
term |= anyElement
```

Syntactically, this extension opens XCLX to embed structured data, such as GML, as sentences or terms, as in line [3] of Code Block 1.6. It also allows the use of an empty element from a foreign namespace as a shortcut for a literal qualified name.

5.1.e Foreign dialects with non-XML-based syntax

It may be desirable to embed sentences in a foreign non-XML-based dialect, such as CLIF or IKL. In particular, this is useful when axiomatically defining semantics by translation. Sentences in a non-XML based syntax will appear as simple content of datatype "xs:string". This extension is enabled by allowing "property" elements to have a `@dialect` attribute and arbitrary simple content. Semantics is inherited from the foreign dialect, and thus must be compatible with CL.

Section 5.2 Methods for extending XCLX Semantics

5.2.a Translation to a semantically-grounded form

This is the method employed in Section 4.2, where expressions are translated either to CLIF or to equivalent XCLX forms whose semantics is already defined. This method is appropriate to define the semantics for core syntax and a few general-purpose extensions, but is not sufficient to deal with user-extensions, which need to be in a machine-readable form.

5.2.b Formal Definition

This method is used when equivalent syntactic forms are not defined elsewhere, and is used in Section 7.2, where the (apparently) novel semantics of the meta-language is introduced.

5.2.c Axiomatic Definition

Using the meta-language (Chapter 7) and **<That/>** (Chapter 6), it is possible to define the semantics of new syntax axiomatically. An example is given in Chapter 10, where the semantics for a simple GML application schema is axiomatically defined.

Chapter 6 XCLX-ab extensions

We have extended the XCLX-ab syntax to include some syntactic forms which do not have a counterpart in CLIF, or are alternate syntactic forms. The extension syntax of XCLX-ab is defined by the Relax NG schema driver file[11], which includes a number of schema modules, each defining a limited extension to the syntax. Because the schema modules were written according to the schema design pattern of (Athan and Boley in press), these extension modules may be freely combined with the core schema modules to generate a customized language with only the desired features.

In Table 2 we define the semantics for the XCLX **<That/>** syntax by translating to IKL (Hayes and Menzel 2006?).

Table 2: Extension syntactic forms of XCLX-ab that have IKL counterparts

	XCLX Expression **E**	IKL Expression T[**E**]	Note
1	**<That> S </That>** where **S** is any XCLX sentence.	**(that T[S])**	

The XCLX extension contains alternate syntactic forms which are "syntactic sugar"; i. e., equivalent to some expression in terms of the syntactic forms already defined in Tables 1-2. Some of these forms are included in Table 3.

Table 3: Alternate syntactic forms of "sweet" XCLX-ab

	XCLX Expression **E**	Equivalent XCLX Expression A(**E**)	Note
1	**<Forall> <binds>** **<Constraint><binds>** **T </binds><op>P</op>** **</Constraint></binds>** **S** **</Forall>**	**<Forall>** **<binds> T </binds>** **<Implies>** **<Atom> <op>P</op> T** **</Atom>** **S** **</Implies>** **</Forall>**	g

11 drivers/xclx-ab-ext_driver.rnc in the attached files.

g It is straightforward to extend the constrained binding syntax for multiple bindings.

	XCLX Expression **E**	Equivalent XCLX Expression A(**E**)	Note
2	<Exists> <binds> <Constraint><binds> **T** </binds><op>**P**</op> </Constraint></binds> **S** </Exists>	<Exists> <binds> **T** </binds> <And> <Atom> <op>**P**</op> **T** </Atom> **S** </And> </Exists>	

The syntax for these and other XCLX extensions are defined in Relax NG schemas in the attached electronic files. While the semantics of some particular extensions has been formally defined in this section, in general the semantics of XCLX extensions may be defined axiomatically using the meta-language described next in Chapter 7.

Chapter 7 XCLX Meta-Language

A meta-language has been defined for XCLX so that the language can be self-extensible. The obvious model for a self-extensible XML-based language is XSLT (Clark 1999b). We first explored using XSLT directly as a meta-language. This proved to be inconvenient as the names of our parameters are XML elements (**<Data/>**, etc) while XSLT is designed to handle parameter names that are character sequences not including the characters [< >]. Therefore we chose to implement a novel set of elements in the XCLX namespace that mimic a subset of XSLT functions.

Section 7.1 XCLX Meta Langage Syntax

We present here an abbreviated version of the meta-language syntax for ease of discussion:

Code Block 7.1: Abbreviated schema for the XCLX meta-language.

```
start = Forall-meta
Forall-meta = element Forall { bname-meta+, sentence}
bname-meta = bname | Data
sentence |= meta-element
meta-element =
  element meta-element { meta-name.choice & meta-term.choice* }
meta-name.choice |=
  attribute meta-name { xsd:QName }
  | meta-name
meta-name = element meta-name { meta-text.choice }
meta-text.choice = (meta-Value-Of | meta-Text)+
meta-term.choice = meta-element | meta-attribute | meta-Namespace
meta-attribute =
  element meta-attribute { meta-name.choice & anyText.choice }
anyText.choice = text | meta-text.choice
meta-Namespace =
  element meta-Namespace { meta-prefix.choice & meta-text.choice }
meta-prefix.choice |=
  attribute meta-name { xsd:NCName }
  | meta-prefix
meta-prefix = element meta-name { meta-text.choice }
meta-Value-Of = element meta-Value-Of { Data }
meta-Text = element meta-Text { text }
```

This abbreviated grammar starts with a **<Forall/>**[1] element with an expanded binding pattern [2] that allows bound **<Data/>** elements. The content model of the **<Forall/>** element is expanded to include a <meta-element/> element [5-6] by a choice combine of the sentence pattern [4]. This element defines the root of the new document after meta-transformation, as described in Section 7.2 below. The name of an element may be stated in an attribute, **@meta-name** [8], if it is a known qualified name. Otherwise, it may be constructed from parameters and text using the **<meta-name/>** element [10].

Simple content is most easily constructed with the **meta-text.choice** pattern [11], which consists of a sequence of **<meta-Text/>** [23] or **<meta-Value-Of/>** [22] elements. An **<meta-Text/>** element contains only text, while an **<meta-Value-Of/>** element may contain only a single **<Data/>** element, which is usually quantified.

A namespace declaration may be constructed with the **<meta-Namespace/>** element [16-17]. Also attributes may be added to an element with the **<meta-attribute/>** element [13-14]. The content models of these two elements are similar, except that for the namespace construction, an **@meta-name** attribute or element corresponds to the prefix, and so must be an xsd:NCName [19], while the name of an attribute must be a qualified name [8].

The full syntax for the XCLX meta-language is given in the Relax NG schema module "**meta_expansion_module.rnc**", in the attached electronic files.

Section 7.2 XCLX Meta Language Semantics

The semantics for the XCLS meta-language is defined in this subsection.

We first define the concept of a meta-variant transformation. Assume ***T*** is an XCLX document containing meta-elements[12]. Further assume *I* is an interpretation of the vocabulary V of ***T*** in the sense of the CL Standard (ISO/IEC 2007). Intuitively, a ***T***-meta-variant transformation *M* realizes one set of possible denotations of universally bound literals (<x:Data datatype="**p:d**">**t**</x:Data> and equivalent forms) that appear in the contents of meta-language elements in the text ***T***, and also transforms those meta-language elements into ordinary XCLX so that the truth-value of the text may be assigned

12 For this discussion, the root element of the document is assumed to be **<x:XCLX/>** and we call **x** the 'root prefix', i. e., prefix of the root element. In general, the root prefix is arbitrary.

by the interpretation *I*.

In particular, let N be the set of (universally) bound literals that appear in meta-language elements in text ***T***. If a name is bound more than once in the text, we may, without loss of generality, change the bound names so that each is bound only once. A ***T***-meta-variant transformation *M* is similar to an N-variant interpretation (defined in (ISO/IEC 2007)) in that the *M* - referent of any name in the set N may take on any value from the vocabulary, subject to constrained bindings[13].

There are two differences between a meta-variant transformation and an N-variant interpretation:

- the meta-variant transformation has a separate set of meta-transformation rules that map character sequences from the text matching certain patterns into new character sequences.
- the meta-variant transformation does not produce a truth value, but instead produces a new XCLX text.

All of the meta-transformation rules, stated in Table 4, generate character sequences. The meta-transformation starts with application to the entire text, *M*(***T***), and continues by recursion until termination with rules from Rows 3, 4, 12-14 or 16.

Table 4: Semantics of the XCLX Meta-language

	XCLX Expression E	***M*(E)**	Note
1	**T_1 T_2... T_n**	*M*(**T_1**) *M*(**T_2 ... T_n**)	a, b
2	**<x:meta-Value-Of>** **T_1 ... T_n** **</x:meta-Value-Of>**	*M*(**T_1 ... T_n**)	a
3	**<x:meta-Value-Of xmlns:p="q">** **<x:Data** **datatype="p:d">t</x:Data>** **</x:meta-Value-Of>**	map_M**(t)**	c

13 Recall from Chapter 6, Table 3, Row 8 that a datatype attribute on a bound variable is an alternate syntactic form for a constrained binding.

a Each **T_i** is a well-formed XML element with name "**meta-Text**" or "**meta-Value-Of**", i = 1, ..., n

b Adjacent character sequences are concatenated.

c Note: the function map_M maps the character sequence **t** into another character sequence in the *lexical* space of the datatype **<q:d/>**, not the *value* space.

	XCLX Expression **E**	M(**E**)	Note
4	**<x:Data datatype="p:d" xmlns:p="q">t</x:Data>**	**<x:Data datatype="p:d" xmlns:p="q" >**map_M**(t)</x:Data>**	c
5	**<x:meta-element>** **S**$_1$ **<x:meta-name>n</x:meta-name> S**$_2$ **</x:meta-element>**	M(**<x:meta-element meta-name="**M**(n)"> S**$_1$ **S**$_2$ **</x:meta-element>**)	
6	**<x:meta-element meta-name="n"> S </x:meta-element>**	M(**<n> S </n>**)	
7	**<n q>** **S**$_1$ **<x:meta-namespace> S**$_2$ **<x:meta-name>m</x:meta-name> S**$_3$ **</x:meta-namespace> S**$_4$ **</n>**	M(**<n q> S**$_1$ **<x:meta-namespace meta-name="**M**(m)"> S**$_2$ **S**$_3$ **</x:meta-namespace> S**$_4$ **</n>**)	e
8	**<n q> S**$_1$ **<x:meta-namespace meta-name="m"> v </x:meta-namespace> S**$_2$ **</n>**	M(**<n q xmlns:**M**(m)="v"> S**$_1$ **S**$_2$ **</n>**)	e
9	**<n q> S**$_1$ **<x:meta-attribute> S**$_2$ **<x:meta-name>m</x:meta-name> S**$_3$ **</meta-attribute> S**$_4$ **</n>**	M(**<n q> S**$_1$ **<x:meta-attribute meta-name="**M**(m)"> S**$_2$ **S**$_3$ **</x:meta-attribute> S**$_4$ **</n>**)	e
10	**<n q> S**$_1$ **<x:meta-attribute meta-name="m"> v </x:meta-attribute> S**$_2$ **</n>**	M(**<n q m="**M**(v)"> S**$_1$ **S**$_2$ **</n>**)	e
11	**<n q> S**$_1$ **... S**$_n$ **</n>**	**<n q>** M(**S**$_1$) . . . M(**S**$_n$)**</n>**	d, e
12	**<x:meta-Text>'</x:meta-Text> <x:meta-Value-Of ckind="/clif#quotedString" > <x:Data>t</x:Data> </x:meta-Value-Of> <x:meta-Text>'</x:meta-Text>**	**T[<x:Data>**map_M**(t)</x:Data>]**	f

d Each **S**$_i$ is either (1) text from the lexical space of "**xs:string**" or (2) a well-formed XML element whose name is (a) "**meta-element**", (b) "**meta-Text**", (c)"**meta-Value-Of**", (d) a non-meta-language qualified name with root prefix, or (e) a qualified name with a non-root prefix.

e **n** is (1) any qualified name with root prefix except for meta-language XCLX element names ("**meta-***"), "**Data**" or "**Forall**", or (2) any qualified name with a non-root prefix.

f T is the translation of XCLX into CLIF defined in Section 4.2, Table 1.

	XCLX Expression **E**	*M*(**E**)	Note

13	`<x:meta-Text>"</x:meta-Text>`
	`  <x:meta-Value-Of`
	`  ckind="/clif#enclosedName"`
	`>`
	`  <x:Data`
	`datatype="xs:token">`**t**`</x:Data>`
	`  </x:meta-Value-Of>`
	`<x:meta-Text>"</x:meta-Text>`
		`T[`
		`<x:Var>`*map*$_M$`(`**t**`)</x:Var>]`	

14	`<x:meta-Value-Of`
	`  ckind="/clif#sentence">`
	`  <x:Data`
	`  datatype="xs:anyComplexType"`
	`  >`**t**`</x:Data>`
	`</x:meta-Value-Of>`
		`T[` *map*$_M$`(`**t**`)]`	

15	`<x:meta-Text>`**T**`</x:meta-Text>`	**T'**	g
16	**T**	**T**	

17	`<x:Forall>`
	`  <x:binds>` **T**$_1$ **n** **T**$_2$ `</x:binds>`
	S `</x:Forall>`
	where **n** is a member of N.
		M(`<x:Forall>`
		`  <x:binds>` **T**$_1$ **T**$_2$ `</x:binds>`
		S `</x:Forall>`)	

18	*M*(`<x:Forall>`
	`  <x:binds>` **T** `</x:binds>`
	S `</x:Forall>`)
	where **T** is a name sequence with no members of N.
		`<x:Forall>`
		`  <x:binds>` **T** `</x:binds>`
		M(**S**) `</x:Forall>`	

19	*M*(`<x:Forall>` `  <x:binds> </x:binds>` **S** `</x:Forall>`)	*M*(**S**)	

Application of the meta-transformation rules until termination removes all meta-language elements with the root prefix from the text. Also all remaining occurrences of names from the set N will have been modified so that they explicitly denote the referents from the meta-transformation *M*. The meta-transformation is the first phase in a multi-phase transformation, and its output is then interpreted as ordinary XCLX in the next phase.

A meta-language text ***T*** with vocabulary V will have a subset $VU_T \subset V$ of universally quantified literals that appear in meta-language elements. A total lexical map N on a set of literals W maps the lexical part of each literal in W into an element of the lexical space

g **T** is any character sequence in the lexical space of the datatype xs:string. **T'** is obtained from **T** by replacing escaped character sequences [`<` `>`] with [< >] in the meta-transformation.

of the datatype of that literal. A meta-language text ***T*** is true in an interpretation *I* of V if and only if

- there exists a set P of meta-variant transformations of *I* such that:
    - For every total lexical map N from VU_T into V, there is some *M* in P such that $map_M(\mathbf{t}) = N(\mathbf{t})$ for every **t** in VU_T.
    - there is a non-empty set P' of all *M* in P where $M^{k(M)}(\boldsymbol{T}) = M^{k(M)+1}(\boldsymbol{T}) = \boldsymbol{T}_M$ for some positive integer k(*M*), and $\boldsymbol{T}_M$ is a valid non-meta-XCLX text;
    - Every member *M* of P' satisfies ***T***, i. e., $I(\boldsymbol{T}_M) = true$.

The evaluation of the meta-transformation requires recursion because there are several (useful) ways to write a meta-language text where the result of applying the meta-transformation rules results in another meta-language text, just as in XSLT it is possible to create a stylesheet that transforms to another XSLT stylesheet:

- use a quantified data variable of type "`xs:anyComplexType`". Certain values of this variable will transform to meta-language elements;
- use a <meta-Text/> element whose contents are escaped meta-language text;
- following the technique used in XSLT, the XCLX namespace is assigned to another prefix, say "`xclx2`", as well as being the namespace of the root prefix. Then <meta-element/> may be used to create new meta-language elements with the "`xclx2`" prefix, and these elements are not processed by rule M6.

Some practical examples of the meta-language are presented in the following subsections, after additional extensions have been introduced.

Chapter 8 Unified Element Frameworks

Unified frameworks in XML schema are not novel; they have appeared in RuleML (Boley, Paschke and Shafiq 2010), XCL1 (ISO/IEC 2007), and MathML (W3C 2010). Although they appear helpful in making the schema extensible, if XSD schemas are to be used then it is necessary to find a common content model appropriate for all values of the modifying attributes or elements.

In this section we briefly, due to length limitations, describe the unified frameworks for XCLX. Modifying attributes are given as choice patterns, such as **`@interp`** |**`@cinterp`**. The second item of the choice always takes a CURIE value. The first item usually takes an IRI, except for **`@datatype`**, which takes a qualified name. All elements belong to the core XCLX namespace and attributes are local (no namespace). The frameworks unify

- names, integrating **`<Data/>`**, **`<Num/>`**, **`<Var/>`** and **`<That/>`** into a single element **`<Name/>`** modified by an optional attribute @**`interp`** |@**`cinterp`**, with a child element **`<lexical/>`** (one required) carrying the required attribute @**`datatype`** | @**`cdatatype`**;

- Boolean sentences, integrating **`<And/>`**, **`<Or/>`**, **`<Implies/>`**, **`<Equiv/>`**, **`<Not/>`** into a single element **`<Boolean/>`** modified by a required attribute **`@kind`** |**`@ckind`**, with child elements **`<term>`** | **`<terms/>|<sentence>|<sentences/>`** (zero to many);

- quantified sentences, integrating **`<Forall/>`**, **`<Exists/>`** into a single element **`<Quantification/>`** modified by a required attribute **`@kind|@ckind`**, and with child elements **`<binds/>`** (one to many), and **`<term>`** | **`<terms/>|<sentence>|<sentences/>`** (zero to many) with a required attribute kind (**`@ckind`**);

- atomic sentences, as a single element **`<AtomicSentence/>`** modified by an optional attribute **`@kind|@ckind`**, with child elements **`<term>`** | **`<terms/>`** (zero to many) with optional **`@kind`** |**`@ckind`** attribute;

- equations, as a single element **`<Equation/>`** modified by an optional attribute **`@kind`** |**`@ckind`** , and with child elements **`<terms>`** | **`<terms>`** (zero to many).

- functions, as a single element **<FunctionalTerm/>** modified by an optional attribute **@kind** |**@ckind**, with child elements **<terms ckind="op"/>** (one to many) and additional **<terms/>** (zero to many) with optional **@kind** |**@ckind** attribute.

The syntax of the unifying frameworks is primarily used in structural axioms defining the semantics for shortcuts, but may also be used directly for user-defined extensions when it is inconvenient to modify the syntax through the schemas. An example using the meta-language to map a shortcut name for resources, **<Var iri=""/>**, into the unified name framework is shown in Code Block 8.1:

Code Block 8.1: Axiomatic Definition of Semantics for **<Var iri=""/>**

```
<Forall>
  <binds><Data datatype="xs:anyURI">z</Data></binds>
  <Equal>
    <Var>
      <meta-attribute meta-name="iri">
        <meta-Value-Of><Data datatype="xs:anyURI"
          >z</Data></meta-Value-Of>
      </meta-attribute>
    </Var>
    <Name>
      <meta-element meta-name="lexical">
       <meta-attribute meta-name="datatype"
          >xs:anyURI</meta-attribute>
        <meta-Value-Of><Data datatype="xs:anyURI"
          >z</Data></meta-Value-Of>
      </meta-element>
    </Name>
  </Equal>
</Forall>
```

A segregated KR language, such as the First-Order Logic branch of Deliberation RuleML (Boley, Paschke and Shafiq 2010), may have many different disjoint naming syntaxes, which can be identified with some portion of the network vocabulary using the XCLX unified name framework.

Several XCLX shortcuts are given here in terms of their corresponding form in the unified framework or an equivalent shortcut already defined:

	XCLX Expression **E**	Equivalent XCLX Expression A(**E**)	Note
1	`<Name cinterp="a:b" xmlns:a="c"> <lexical datatype="p:q" >m</lexical> </Name>`	`<Name interp="cb"> <lexical datatype="p:q" >m</lexical> </Name>`	
2	`<Var i iri="q"/>`	`<Name i> <lexical datatype="xs:anyURI" >q</lexical> </Name>`	a
3	`<Var i cri="p:q" xmlns:p="n"/>`	`<Var i iri="nq"/>`	a
4	`<p:q i xmlns:p="n"/>`	`<Name i xmlns:p="n"> <lexical datatype="xs:QName" >p:q</lexical> </Name>`	a
5	`<Data datatype="p:q">m</Data>`	`<Name cinterp="/nm#lit"> <lexical datatype="p:q" >m</lexical> </Name>`	
6	`<Data datatype="p:q">m</Data>`	`<Expr> <op> <p:q/> </op> <Data>m</Data> </Expr>`	
7	`<Num>m</Num>`	`<Data datatype= "xs:nonNegativeInteger" >m</Data>`	
8	`<Data>m</Data>`	`<Data datatype="xs:string">m</Data>`	
9	`<Var i>m</Var>`	`<Var i datatype="xs:token">m</Data>`	a
10	`<That>m</That>`	`<Name cinterp="/nm#that"> <lexical datatype="xs:anyComplexType" >m</lexical> </Name>`	
11	`<Expr> <op> <Var iri="q"/> </op> <Expr> <op> <p:q/> </op> <Data>m</Data> </Expr> </Expr>`	`<Name interp="q"> <lexical datatype="p:q" >m</lexical> </Name>`	

a where **i** is any attribute assignment except `@datatype`.

Chapter 9 Modules and Context

Section 9.1 Modules

XCLX implements modules following the lead of CLIF (ISO/IEC 2007):

	XCLX Expression **E**	Equivalent XCLX Expression A(**E**)	Note
1	`<Module name="u">` `<excludes>`**L**$_1$ … **L**$_n$ `</excludes>`**S**`</Module>`	`<Not> <Atom>` `<op><Var iri="u"/></op>` **L**$_1$ `</Atom> </Not` … `<Not> <Atom>` `<op><Var iri="u"/></op>` **L**$_n$ `</Atom> </Not>` **S'**	a

Modules are required in the implementation of segregated dialects, such as FOL RuleML (Boley, Paschke and Shafiq 2010), where it is necessary to specify that certain names are non-discourse names.

Section 9.2 Context

No additional syntax or semantics is required to implement context in the manner of IKL. To assert a text in a context "**c**", it is sufficient to call each sentence **S** of the text within an atomic sentence using the "**that**" syntax;

Code Block 9.1: XCLX contextual sentence using the IKL style.

```
<Atom> <op><Var iri="c"/></op> <That> S </That> </Atom>
```

To also restrict the vocabulary, as in the contextual approach of Local Models Semantics (Ghidini and Giunchiglia 2001), one may use the **<Module/>** construct to restrict the local universe of discourse for entire texts, or the **@interp** (**@cinterp**) attribute on **<That/>**, to restrict the local universe of discourse for individual sentences.

We do not implement the contextual approaches based on modal operators, nor do we

a **S'** is obtained from **S** by replacing any bindings of names **L**$_1$ … **L**$_n$ with bindings constrained by `<Var iri="u"/>`.

introduce the ICL pragma notation (IKRIS 2007a) into XCLX at this time. The syntax of such an extension is straightforward, but the semantics is more challenging, as there is not an active language with formal semantics to reference.

Chapter 10 GML Semantics in XCLX

We will now demonstrate how to express the semantics of GML structured data in XCLX. The GML in Code Block 10.1 is an excerpt of Code Block 1.4 - the method we describe here can be extended to apply to instances of arbitrary GML application schemas, to structured data that do not follow a fully striped XML data model, and even to non-XML formats such as JSON (Crockford 2002).

The syntax of structured data may be provided by XML schema, but that is not sufficient to define its semantics. If we can build an implication in the meta-language that maps an arbitrary instance of the schema to an XCLX sentence, then the semantics will be determined.

Code Block 10.1: Structured data containing geographical information.

```
1  <nws:NewsItem gml:id="G3">
2    <nws:location>
3      <gml:Point gml:id="G4" srsDimension="2"
4        srsName="urn:ogc:def:crs:EPSG:6.6:4326">
5        <gml:pos>29.5 -123.05</gml:pos>
6      </gml:Point>
7    </nws:location>
8    <nws:reporterId>R1</nws:reporterId>
9    <nws:eventDate>2011-08-16T14:05:38</nws:eventDate>
10   <nws:byLine>Tara Athan</nws:byLine>
11   <nws:details>Blah blah.</nws:details>
12   <nws:image mimeType="image/jpeg"
13     url="http://www.mynewsservice.org/jpeg-ar/324589.jpg"/>
14 </nws:NewsItem>
```

We utilize the **`gml:id`** attribute to expand the nested GML into stand-alone "objects". We illustrate for the "property" element "**`<nws:location/>`**". The more general axiom, which holds for arbitrary element names whose namespace belongs to a specified set, is given in the attached electronic file, as "**`meta-gml-10-2a.xclx`**".

Code Block 10.2: Axiom for converting nested GML to stand-alone features.

```
<Forall> <binds>
  <Data datatype="xs:NCName">ni_id</Data>
  <Data datatype="xs:anyComplexType"><ni_rest_content/></Data>
  <Data datatype="xs:NCName">gp_id</Data>
  <Data datatype="xs:anyComplexType"><gp_content/></Data> </binds>
  <Implies>
    <nws:NewsItem>
      <meta-attribute meta-name="gml:id">
        <meta-Value-Of><Data datatype="xs:NCName"
          >ni_id</Data></meta-Value-Of> </meta-attribute>
      <nws:location>
        <gml:Point>
          <meta-attribute meta-name="gml:id">
            <meta-Value-Of><Data datatype="xs:NCName"
              >gp_id</Data></meta-Value-Of>
          </meta-attribute>
          <meta-Value-Of>
            <Data datatype="xs:anyComplexType"><gp_content/></Data>
          </meta-Value-Of>
        </gml:Point>
      </nws:location>
      <meta-Value-Of> <Data datatype="xs:anyComplexType">
            <ni_rest_content/></Data> </meta-Value-Of>
    </nws:NewsItem>
    <Exists> <binds> <Var>news_item</Var>
      <Var>point</Var> </binds> <And>
        <Equal>
          <Var>news_item</Var>
          <Var> <meta-attribute meta-name="iri">
            <meta-Text>#</meta-Text>
            <meta-Value-Of><Data datatype="xs:NCName"
              >ni_id</Data></meta-Value-Of>
          </meta-attribute> </Var> </Equal>
        <Equal>
          <Var>point</Var>
          <Var> <meta-attribute meta-name="iri">
            <meta-Text>#</meta-Text>
            <meta-Value-Of><Data datatype="xs:NCName"
              >gp_id</Data></meta-Value-Of>
          </meta-attribute> </Var> </Equal>
        <Atom><op><nws:NewsItem/></op>
          <Var>news_item</Var>
        </Atom>
        <Atom><op><nws:location/></op>
          <Var>news_item</Var> <Var>point</Var>
        </Atom>
        <gml:Point> <meta-attribute meta-name="gml:id">
          <meta-Value-Of><Data datatype="xs:NCName"
            >gp_id</Data></meta-Value-Of>
        </meta-attribute>
          <meta-Value-Of> <Data datatype="xs:anyComplexType">
              <gp_content/></Data> </meta-Value-Of>
        </gml:Point>
      </And> </Exists>
  </Implies>
</Forall>
```

XLCX and GML

We now discuss the axiom presented in Code Block 10.2 . The outermost element is a universal quantification [1] with bindings to "literal" datatyped names contained in meta-language elements. The names are:

- (**ni_id**), the GML identifier for the **<NewsItem/>** feature [2];
- (**ni_rest_content**), the rest of the content of **<NewsItem/>**, other than the child element we will be expanding [3];
- (**gp_id**), the GML identifier for the **<gml:Point/>** geometry object [4];
- (**gp_content**), all of the contents of **<gml:Point/>** including meta-language elements to describe the attributes [5];

The body of the quantifier is an implication [6], whose premise [7-24] translates to the original GML in Code Block 10.1 and similar instances. We don't spend a lot of effort constraining the values of the character sequences that will be processed by the meta-transformation, as we cannot possibly specify conditions on these variables that will guarantee valid instances of the GML application schema, and the meta-language semantics ignores any meta-variants that transform to invalid syntax.

The conclusion of the implication [25-54] is an existentially- quantified conjunction. The quantification is introduced as a convenience, so that we may "declare" some abbreviations [25-26] for the features that are referenced by their GML identifiers.

Conjunction [24-53] contains five sentences. The first two [27-40] just equate the bound variables with the features referenced by GML identifiers. The third sentence [41-43] expresses that the object "**ni_id**" is a member of the "class" **<nws:NewsItem/>**[14]. The fourth sentence [44-46] expresses the connection between the news item and the geometry object through the relation **<nws:location/>**. The fifth sentence [47-53] contains the specifications of the geometry object as stand-alone GML. This GML sentence may be matched to the premise of another, similar, axiom for GML objects with simple content to further extend the implication tree.

14 In CL, the more precise wording is that **<Var>ni_id</Var>** denotes an object in the extension of the relation associated with the object denoted by **<nws:NewsItem/>**, with the qualified name appropriately resolved.

Chapter 11 Conclusions

We have created a Common Logic extension, XCLX that allows embedded structured data such as GML. XCLX contains a subset that is equivalent to the CL dialect CLIF and syntactically is nearly identical to a subset of RuleML. An XCLX extension adopts the semantics of the IKL "**`that`**" structure, enabling named propositions and contextual statements. A meta-language syntax and semantics for XCLX has been developed, which allows the semantics of embedded GML and some extensions to be axiomatically defined. Further extensions include unified frameworks for several syntactic categories, including names. By exploiting a number of features of XML and supporting languages such as XSLT, the XCLX language provides an enhanced level of interoperability between general-purpose KR languages and XML-based structured data.

The XCLX language as presented here is a draft that may be revised after proposal to the Common Logic community. In particular, URIs for namespaces and the values of modifying attributes need to be finalized. Further work may include a reasoner that can process the new language, and completion of a library of axioms for the semantics of GML.

References

ATHAN, T. and H. BOLEY. in press. Design and Implementation of Highly Modular Schemas for XML: Customization of RuleML in RelaxNG. *In:* F. OLKEN, M. PALMIRANI and D. SOTTARA, eds. *Rule-Based Modeling and Computing on the Semantic Web.* RuleML 2011 - America, LNCS 7018.

BECKETT, D. and S. MCBRIDE. 2004. *RDF/XML Syntax Specification (Revised)* [online]. [Accessed Dec 5, 2009]. Available from: http://www.w3.org/TR/REC-rdf-syntax/.

BERNERS-LEE, T., R. FIELDING and L. MASINTER. 2005. *Uniform Resource Identifier (URI): Generic Syntax* [online]. [Accessed 2011-07-16]. Available from: http://tools.ietf.org/html/rfc3986.

BIRBECK, M. and S. MCCARRON. 2010. *CURIE Syntax 1.0* [online]. [Accessed 2011-08-11]. Available from: http://www.w3.org/TR/curie/.

BOLEY, H. 2003a. Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms . *In:* M. SCHRÖDER and G. WAGNER, eds. *Rules and Rule Markup Languages for the Semantic Web.* Springer Berlin / Heidelberg, pp.1-16.

BOLEY, H. 2003b. The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. *In:* O. BARTENSTEIN, U. GESKE, M. HANNEBAUER and O. YOSHIE, eds. *Web Knowledge Management and Decision Support.* Springer Berlin / Heidelberg, pp.5-22.

BOLEY, H. 2010. *[CL] Proposals for XCL 2.0* [online]. [Accessed 2011-09-03]. Available from: http://philebus.tamu.edu/pipermail/cl/2010-October/002179.html.

BOLEY, H., A. PASCHKE and O. SHAFIQ. 2010. RuleML 1.0: The Overarching Specification of Web Rules. *In:* M. DEAN, J. HALL, A. ROTOLO and S. TABET, eds. *Semantic Web Rules.* Springer Berlin / Heidelberg, pp.162-178.

XLCX and GML

BOOLE, G. 1854. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities.* London: Walton and Maberly.

BRICKLEY, D. 2004. *RDF: Understanding the Striped RDF/XML Syntax* [online]. [Accessed 2011-09-03]. Available from: http://www.w3.org/2001/10/stripes/.

CLARK, J. 1999a. *XML Namespaces* [online]. [Accessed 2011-07-16]. Available from: http://www.jclark.com/xml/xmlns.htm.

CLARK, J. 1999b. *XSL Transformations (XSLT) Version 1.0* [online]. [Accessed 2011-08-23]. Available from: http://www.w3.org/TR/xslt.

CLARK, J. 2009. *Jing-Trang Schema validation and conversion based on RELAX NG* [online]. [Accessed 2011-08-01]. Available from: http://code.google.com/p/jing-trang/.

CROCKFORD, D. 2002. *Introducing JSON* [online]. [Accessed 2011-09-11]. Available from: http://json.org/.

DECKER, S., S. MELNIK, F. VAN HARMELEN, D. FENSEL, M. KLEIN, J. BROEKSTRA, M. ERDMANN and I. HORROCKS. 2000. The Semantic Web: the roles of XML and RDF. *Internet Computing, IEEE* **4**(5), pp.63-73.

DUERST, M. and M. SUIGNARD. 2005. *Internationalized Resource Identifier (URI)* [online]. [Accessed 2011-07-16]. Available from: http://tools.ietf.org/html/rfc3987.

FONSECA, R. L. and E. G. LLANO. 2011. Automatic Representation of Geographical Data from a Semantic Point of View through a New Ontology and Classification Techniques. *Transactions in GIS,* **15**(1), pp.61-85.

GAO, S., H. BOLEY, D. MIOC, F. ANTON and X. YI. 2009. Geospatial-Enabled RuleML in a Study on Querying Respiratory Disease Information. *Lecture Notes in Computer Science.* Springer, pp.272-281.

GENESERETH, M. R. 1998. *Knowledge Interchange Format* [online]. [Accessed 2011-08-25]. Available from: http://logic.stanford.edu/kif/dpans.html.

GEOSERVER. 2010. *Welcome - Geoserver* [online]. [Accessed 2010-05-02]. Available

XLCX and GML

from: http://geoserver.org/display/GEOS/Welcome.

GHIDINI, C. and F. GIUNCHIGLIA. 2001. Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. *Artificial Intelligence,* **127**(2), pp.221--259.

GOOGLE. 2011. *Google Earth Home Page* [online]. [Accessed 2011-09-09]. Available from: http://earth.google.com.

GUHA, R. and J. MCCARTHY. 2003. Varieties of contexts. *In:* P. BLACKBURN, C. GHIDINI, R. M. TURNER and F. GIUNCHIGLIA, eds. *Modeling and Using Context, Proceedings.* Berlin: Springer-Verlag Berlin, pp.164-177.

HAROLD, E. R. 2004. *RELAX NG with custom datatype libraries* [online]. [Accessed 2011-07-16]. Available from: http://www.ibm.com/developerworks/xml/library/x-custyp/.

HAYES, P. and C. MENZEL. 2006? *IKL Specification Document* [online]. [Accessed 2011-03-05]. Available from: http://www.ihmc.us/users/phayes/ikl/spec/spec.html.

HAYES, P. J. 2004. *RDF Semantics* [online]. [Accessed Jan 19, 2011]. Available from: http://www.w3.org/TR/rdf-mt/.

HAYES, P. J. 2006? *IKL Guide* [online]. [Accessed 2011-03-05]. Available from: http://www.ihmc.us/users/phayes/IKL/GUIDE/GUIDE.html.

HIRTLE, D., T. DEMA and H. BOLEY. 2006. *The Modularization of RuleML* [online]. [Accessed 2011-02-01]. Available from: http://ruleml.org/modularization/.

IKRIS. 2007a. *ICL Specification* [online]. [Accessed 2011-08-23]. Available from: http://nrrc.mitre.org/NRRC/Docs_Data/ikris/ICL_spec.pdf.

IKRIS. 2007b. *ICL User's Guide* [online]. [Accessed 2011-09-01]. Available from: http://nrrc.mitre.org/NRRC/Docs_Data/ikris/ICL_guide.pdf.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. 2008. *Current defect report for ISO/IEC 24707:2007 as of 03-Feb-08.* [online]. [Accessed 2010-05-25]. Available from: http://common-logic.org/24707-defect-report.pdf.

ISO. 2005. 19101. *Geographic information. Reference model* Geneva: International

Organization for Standardization.

ISO/IEC. 2007. 24707. *Information technology — Common Logic (CL): a framework for a family of logic based languages*. Geneva: International Organization for Standardization.

ISO/IEC. 2008. 19757-2. *Document Schema Definition Language (DSDL) Part 2: Regular-grammar-based validation - RELAX NG*. Geneva: International Organization for Standardization.

KEßLER, C., M. RAUBAL and C. WOSNIOK. 2009. Semantic Rules for Context-Aware Geographical Information Retrieval. *In:* P. BARNAGHI, K. MOESSNER, M. PRESSER and S. MEISSNER, eds. *Smart Sensing and Context.* Springer Berlin / Heidelberg, pp.77-92.

KORTA, K. and J. PERRY. 2011. *Pragmatics* [online]. [Accessed 2011-08-16]. Available from: http://plato.stanford.edu/entries/pragmatics/.

KOTTMAN, C. and C. REED. 2009. *The OpenGIS® Abstract Specification; Topic 5: Features; OGC Document # 08-126* [online]. [Accessed 2011-08-26]. Available from: http://portal.opengeospatial.org/files/?artifact_id=29536.

MARSH, J., D. ORCHARD and D. VEILLARD. 2006. *XML Inclusions (XInclude) Version 1.0 (Second Edition)* [online]. [Accessed 2011-08-23]. Available from: http://www.w3.org/TR/xinclude/.

MCCARTHY, J. 1979. *History of Lisp* [online]. [Accessed 2011-08-25]. Available from: http://www-formal.stanford.edu/jmc/history/lisp/lisp.html.

MCCARTHY, J. and P. J. HAYES. 1969. Some Philosophical Problems from the Standpoint of Articial Intelligence. *In:* B. MELTZER and D. MICHIE, eds. *Machine Intelligence 4,* . Edinburgh University.

OGC. 2007a. 07-147r2. *OGC KML* . Open Geospatial Consortium.

OGC. 2007b. 07-036 *OpenGIS Geography Markup Language (GML) Encoding Standard* Open Geospatial Consortium.

OGC. 2009. 08-126. *OpenGIS Abstract Specification; Topic 5: Features* Open Geospatial

Consortium.

OGC. 2010a. 08-062r4 *OGC Reference Model* Open Geospatial Consortium.

OGC. 2010b. 09-025r1. *Open GIS Web Feature Service 2.0 Interface Standard* Open Geospatial Consortium.

OGC. 2011. 10-100r3. *Geography Markup Language (GML) simple features profile (with technical note)* Open Geospatial Consortium.

PEASE, A. 2009. *Suggested Upper Merged Ontology (SUMO)* [online]. [Accessed Oct 30, 2009]. Available from: http://www.ontologyportal.org/.

REITER, R. 1978. On closed world data bases. *In:* H. GALLAIRE and J. MINKER, eds. *Logic and Data Bases*. New York: Plenum, pp.119–40.

ROBINSON, W. S. 1950. Ecological Correlations and the Behavior of Individuals. *American Sociological Review,* **15**(3), pp.351–357.

SYNCRO SOFT LTD. 2011. *oXygen XML editor* [online]. [Accessed 2011-09-01]. Available from: http://www.oxygenxml.com/.

VLIST, E. V. D. 2003. *RELAX NG*. O'Reilly.

W3C. 2000. *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)* [online]. [Accessed 2011-09-03]. Available from: http://www.w3.org/TR/xhtml1/.

W3C. 2004a. *W3C Recommendation: OWL Web Ontology Language* [online]. [Accessed 2009-12-05]. Available from: http://www.w3.org/TR/owl-features/.

W3C. 2004b. *XML Schema Part 1: Structures* [online]. [Accessed 2010-12-08]. Available from: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

W3C. 2004c. *XML Schema Part 2: Datatypes Second Edition* [online]. [Accessed 2011-02-03]. Available from: http://www.w3.org/TR/xmlschema-2/.

W3C. 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. [Accessed 2010-12-08]. Available from: http://www.w3.org/TR/REC-xml/.

W3C. 2010. *Mathematical Markup Language (MathML) Version 3.0: Appendix A.*

Parsing MathML [online]. [Accessed Jan 27, 2011]. Available from: http://www.w3.org/TR/MathML3/appendixa.html.

W3C OWL WORKING GROUP. 2009. *OWL 2 Web Ontology Language: Document Overview* [online]. [Accessed Dec 10, 2009]. Available from: http://www.w3.org/TR/owl2-overview/.

WORLD METEOROLOGICAL ORGANIZATION. 2004. *ISO 19100 SERIES OF GEOGRAPHIC INFORMATION STANDARDS* [online]. [Accessed 2011-08-26]. Available from: http://www.wmo.int/pages/prog/www/ISS/Meetings/ITT-FWIS_Geneva2004/5(2)_ISO.doc.

Appendix A. Table of Acronyms

Acronym	Meaning
ANTLR	ANother Tool for Language Recognition
CGIF	Conceptual Graph Interchange Format
CL	Common Logic
CLIF	Common Logic Interchange Format
CURIE	Compact URI
DOM	Document Object Model
DTD	Document Type Definition
EBNF	Extended Backus-Naur Format
GML	Geographic Markup Language
GPS	Global Positioning System
ICL	IKRIS Context Language
IKL	Interoperable Knowledge representation Language
IRI	Internationalized Resource Identifier
ISO	International Organization for Standardization
KIF	Knowledge Interchange Format
KR	Knowledge Representation
Lisp	LISt Processing
OGC	Open Geospatial Consortium
OIL	Ontology Interchange Language
OWL	Web Ontology Language
RDF	Resource Description Framework
RuleML	Rule Markup Language
SGML	Standard Generalized Markup Language
SRS	Spatial Reference System
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
wff	well-formed formula
WFS	Web Feature Service
XCL	eXtended Common Logic Markup Language

XLCX and GML

Acronym	Meaning
XCLX	XML-based Common Logic Extensions
XCLX-ab	XCLX-abbreviated form
xHTML	extensible HyperText Markup Language
XInclude	XML Inclusions
XML	Extensible Markup Language
XSD	XML Schema Definition Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations