W3C Incubator Group Report

Common Web Language

W3C Incubator Group Report 31 March 2008

This version:
http://www.w3.org/2005/Incubator/cwl/XGR-cwl-20080331/
Latest version:
http://www.w3.org/2005/Incubator/cwl/XGR-cwl/
Previous version:
This is the first public report.
Editor:
Hiroshi Uchida
Authors:
Hiroshi Uchida, Institute of Semantic Computing
Toshio Yokoi, Institute of Semantic Computing
Meiying Zhu, Institute of Semantic Computing
Nobuo Saito, Keio University
Vahan Avetisyan, UNL Armenian Language Centery
Contributors:
Hiroshi Yasuhara, Institute of Semantic Computing
Also see Acknowledgements

Abstract

This is a report of the W3C Common Web Language Incubator Group (CWL-XG) as specified in the Deliverrables of its charter.

In this report we define the Common Web Language (CWL) Specifications and introduce various forms of representation of CWL and its platform to work with CWL.

Specifically the report:

- defines, the CWL Archtecture,
- presents the CWL specification,
- discusses the merit to have various forms of representation,
- describes, how to use CWL using CWL platform
- proposes, the use for CWL for multilingualism of web.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of Final Incubator Group Reports is available. See also the W3C technical reports index at http://www.w3.org/TR/.

This document was developed by the W3C Common Web Language Incubator Group, part of the W3C Incubator Activity.

Publication of this document by W3C as part of the W3C Incubator Activity indicates no endorsement of its content by W3C, nor that W3C has, is, or will be allocating any resources to the issues addressed by it. Participation in Incubator Groups and publication of Incubator Group Reports at the W3C site are benefits of W3C Membership.

Incubator Groups have as a goal to produce work that can be implemented on a Royalty Free basis, as defined in the W3C Patent Policy. Participants in this Incubator Group have made no statements about whether they will offer licenses according to the licensing requirements of the W3C Patent Policy for portions of this Incubator Group Report that are subsequently incorporated in a W3C Recommendation.

Table of Contents

1. Introduction

The CWL, a common web language for humans and computers, must solve the following two big problems exist in the present web world. One is the language barriers in the web, and another is lacking of machine understandability for the contents in the web.

Language barrier:
Currently almost all web pages are written in English. It is convenient for English speaking people but it is not for non-English speaking people, and those people are the majority in the world. Those people cannot get information easily, because it is not written in their mother tongue. Recently machine translation facilities are equipped in the web, but it is not the solutions. Machine translation has a problem of quality and coverage of languages.

Machine Understandability:
HTML tags give information on structure of web documents, but they do not give semantic information on each words nor sentences in documents. It means HTM tags information is insufficient to intellectually utilize contents of web pages. The RDF and OWL have a framework to give semantic information but they do not have standard vocabulary to describe web contents.

1.1 Objective of CWL

Objectives of CWL is for exchanging information through the web and also for enabling computers to process information semantically. The CWL allows people to describe contents and meta-data of web pages written in natural languages and also allows realizing a language barrier free world in the web and it will also enable computers to extract semantic information and knowledge from web pages accurately.

1.2 Requirement for CWL

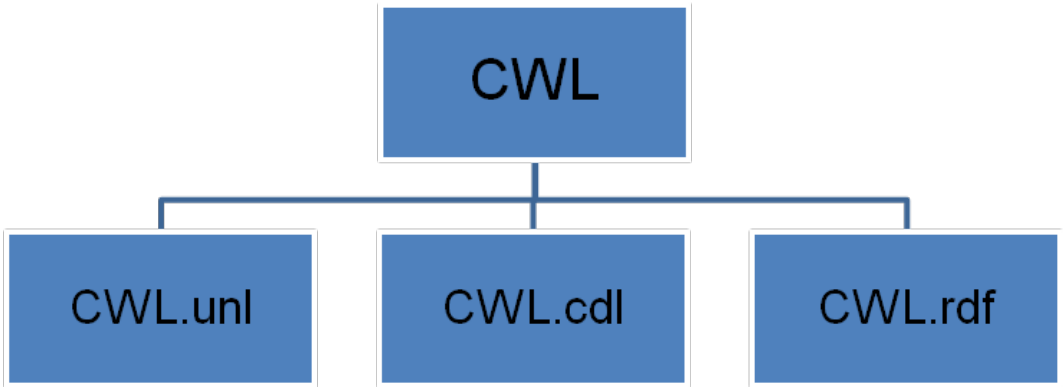
The requirements for CWL to achieve the above objectives are the followings.

1. To be independent from any natural languages and shall enable users to develop conversion systems between CWL and each natural language easily.
2. Different from natural languages, CWL is a unambiguous formal language playing the same role of natural languages for humans.
3. Concepts included in any natural language can be the vocaburary of CWL.
4. Surface structure and semantic structure of original document must be expressed in CWL.

2. CWL Architecture

The CWL is a graph language of semantic hyper directed graph, a node represents a concept, an arc represents a relation between nodes and a node can be annotated by attributes. This CWL can be expressed in three languages UNL, CDL and RDF/OWL. The same information in CWL can be described in each language but in different manner. The CWL.unl is a language based on UNL. The CWL.cdl is a language based on the CDL. The CWL.rdf is a language based on RDF/OWL.

Three different types of representations of CWL allow different way of treatment of the same information described. The CWL.unl is for multilingualism, The CWL.cdl has compatibility with semantic computing systems for semantic computing, The CWL.rdf is for working with various data navigation and aggregation systems (like SPARQL).



Compatibility among expressions in UNL (CWL.unl), CDL (CWL.cdl), and RDF is guaranteed. Followings are the expression of each representation for the sentence `I purchased a computer yesterday.'.

CWL.unl expression

```
{unl} //Table Form of UNL expression
agt(purchase(icl>buy(agt>person,obj>thing)).@entry.@past), I)
obj(purchase(icl>buy(agt>person,obj>thing)).@entry.@past),computer(icl>machine))
tim(purchase(icl>buy(agt>person,obj>thing)).@entry.@pst), yesterday(icl>day))
{/unl}
```

CWL.cdl expression

```
{#S Situation;
  {#A Event tmp='past';
    {#A1 purchase (icl>buy(agt>person,obj>thing) ;}
    {#A2 I ral='def';}
    {#A3 computer(icl>machine) ral='def';}
    {#A4 yesterday(icl>day) ral='def';}
    [#A1 cdd.nl#agt #A2]
    [#A1 cdd.nl#obj #A3]
    [#A1 cdd.nl#tim #A4]
  }
}
```

CWL.rdf expression

```
RDF// N-Triples representation: Subject Property Object"."
#S rdf:type Situation.
#A rdf:type Event.
#S hasComplexEntity #A.
#A hasElementalEntity #A1.
#A hasElementalEntity #A2.
#A hasElementalEntity #A3.
#A hasElementalEntity #A4.
#A1 rdf:type purchase(icl>buy(agt>person,obj>thing).
#A2 I rdf:type I.
#A2 I ral 'def'.
#A3 rdf:type computer(icl>machine).
#A3 ral 'def'.
#A4 rdf:type yesterday(icl>day).
#A4 ral 'def'.
#A1 agt #A2.
#A1 obj #A3.
#A1 tim #A4.
#A tmp 'past'.
```

2.1 CWL

The CWL is designed to be used to describe meta-data and contents of web pages for breaking language barriers and enable computers to process web information semantically. The CWL is a graph language of extended semantic hyper directed graph, a node represents a concept, an arc represents a relation between nodes and a node can be annotated by attributes. Every node including nodes in hyper nodes can have relation with any other nodes.

A concept can be expressed in any kind of a morpheme, a word, a phrase, a clause or a sentence of any language and a Universal Word developed by the UNDL Foundation. A concept (character strings) express the meaning/concept that a a morpheme, a word, a phrase, a clause or a sentence can express in each language.

The meaning of a concept in CWL depend on the context. It means the meaning of a concept in CWL is restricted by relations with other concepts linked with arcs going in and out.

The following is a CWL.unl graph representation for the sentence `Long ago, in the city of Babylon, people begun to build a huge tower, which seemed about to reach the heavens.'

```
begin.entry.past-- tim->long ago
    --plc->city.def
    --agt->people.def
    --obj->build.past--obj-> tower<- aoj--huge<- seem.past
                     --obj->reach.bigin.soon--obj->tower
                     --gol->heaven.def.pl
```

Compatibility among expressions in UNL (CWL.unl), CDL (CWL.cdl), and RDF is guaranteed. Followings are the expression of each representation for the sentence `I purchased a computer yesterday.'.

```
{cwl.unl}
tim(begin.@entry.@past,long ago)
mod(city.@def,Babylon)
plc(begin.@entry.@past,city.@def)
agt(begin.@entry.@past,people.@def)
obj(begin.@entry.@past,build.@past)
agt(build,people.@def)
obj(build,tower)
aoj(huge,tower)
aoj(seem.@past,tower)
obj(seem.@past,reach.@begin.@soon)
obj(reach.@begin.@soon,tower)
gol(reach.@begin.@soon,heaven.@def.@pl)
{/cwl.unl}
```

Relations constitute syntax of the CWL. They expresses objectivity together with UWs by describing how concepts(UW) constitutes a sentence related each other. We adopt the relation set of the UNL relation set for the basis of the relation set of CWL. Psudo relations will be added as the need arises.

There are 41 relations, such as "agt", "aoj", "bas", "ben", "cag", "cao", "cnt", "cob", "con", "coo", "dur", "equ", "fmt", "frm", "gol", "icl", "ins", "int", "iof", "man", "met", "mod", "nam", "nxt", "obj", "or", "per", "plc", "plf", "plt", "pof", "pos", "ptn", "pur", "qua", "ref", "rsn", "scn", "seq", "shd", "src", "tim", "tmf", "tmt", "to", and "via".

Attributes describe mainly subjectivity. We also adopt the attribute set of the UNL attribute set as a basis of the attribute set of the CWL. Attributes will be added as the need arises. Attributes are categorized into following 7 groups.at this moment.

Times with respect to the writer
@past, @present, @future
Writer's view on aspects of event
@begin, @complete, @continue, @custom, @end, @experience, @progress, @repeat, @state
Writer's view of emphasis, focus and topic
@contrast, @emphasis, @entry, @qfocus, @theme, @title, @topic
Writer's attitudes
@affirmative, @confirmation, @exclamation, @humility, @imperative, @interrogative, @invitation, @polite, @request, @respect, @vocative
Writer's feelings and judgments
@ability, @get-benefit, @give-benefit, @conclusion, @consequence, @sufficient, @consent, @dissent, @grant, @grant-not, @although, @discontented, @expectation, @wish, @insistence, @intention, @want, @will, @need, @obligation, @obligation-not, @should, @unavoidable, @certain, @inevitable, @may, @possible, @probable, @rare, @unreal, @admire, @blame, @contempt, @regret, @surprised, @troublesome
Writer's view of reference
@generic, @def, @indef, @not, @ordinal
Describing logical characters and properties of concepts
@transitive, @symmetric, @identifiable, @disjointed
Modifying attribute on aspect
@just, @soon, @yet, @not
Attribute for Convention
@passive, @pl, @angle_bracket, @brace, @double_parenthesis,@double_quote, @parenthesis, @single_quote, @square_bracket

2.1.1 Vocabulary of CWL

A concept can be expressed in any kind of a morpheme, a word, a phrase, a clause or a sentence of any language and a Universal Word developed by the UNDL Foundation. A concept (character strings) express the meaning/concept that a morpheme, a word, a phrase, a clause or a sentence can express in each language.

The CWL vocaburary is defined as the CWL ontology. Every concept must be defined in the CWL ontroly. The CWL ontology gives the linguistic knowledge of the CWL. The following hierarchy of concepts is a part of the top ontology.

```
concept
    nominal copncept
        thing
            abstract thing
                attribute
                    quality
                    feature
                event
                    action
                        mental action
                        physical action
                        process
                    phenomenon
                        mental phenomenon
                        physical phenomenon
                        change
                        process
                information
                    statement
                    description
                quantity
                rule
                state*
                    condition
                    mental state
                    physical state
                    situation
                way
                    arrangement
                    behavior
```

```
                    manner
                    method
            attributive thing
                group
                    group(icl>volitional thing)
                    set
            concrete thing
                living thing
                    human
                    animal
                    plant
                natural world
                substance
                ...
            functional thing
                facilities
                symbol
                tool
                ...
            volitional thing
                animal
                group
                human
                ...
            place
                area
                relative place
                ...
            time
                period
                ...
    predicative concept
        do
            act
            express
                state
            get
                obtain
            give
                provide
            make
                produce
            take
            change
            move
            put
            mentally do
            physically do
            do(agt>thing)
            do(agt>thing,obj>thing)
            ...
            treat
            deal
            perform
        occur
            become
            happen
                change
                move
                mentally happen
                physically happen
                ...
            occur(obj>thing)
            occur(gol>thing,obj>thing)
            ...
        be
            be(aoj>thing,obj>thing)
            be(aoj>thing)
            ...
    attributive concept
        (qua<thing)    :quantifier
        (mod<thing)    :modifier
        ...
    adverbial concept
        (qua>quantifier)
        (man<predicative concept)
            how
```

The CWL ontology also defineds every possible relations between concepts.

2.1.2 Relations

There are many factors to be considered in choosing an inventory of relations between concepts. Different factors taken into account in choosing the relations lead to different sets of the relations. We aopt the relation set of the UNL. The UNL relations are selected basically according to the following principles:

Principles of Relation

PRINCIPLE 1 : NECESSARY CONDITION

When a concept has relations between more than one other concepts, each relation label should be set so as to be able to identify each relation on the premise that there is enough knowledge about the concept of each concept expressed.

PRINCIPLE 2 : SUFFICIENT CONDITION

When there are relations between concepts, each relation label should be set so as to be able to understand the role of each concept only by referring to the relation label.

```
Relation
    predicative relation
        agt(agent)
        aoj(thing with attribute)
          cag(co-agent)
          cao(co-thing with attribute)
```

```
        ptn(partner)
        ben(beneficiary)
        cob(affected co-thing)
        obj(affected thing)
        opl(affected place)
        ins(instrument)
        met(method or means)
        man(manner)
        plc(place)
        plf(initial place)
        plt(final place)
        scn(scene)
        gol(goal, final state)
        src(source, initial state)
        via(intermediate place or state)
        dur(duration)
        tim(time)
        tmf(initial time)
        tmt(final time)
  inter-concept relation
        and(conjunction)
        or(disjunction, alternative)
        fmt(range/from-to)
        frm(origin)
        to(destination)
        equ(equivalent)
        icl(included/a kind of)
        iof(an instance of)
  inter-event relation
      con(condition)
        coo(co-occurrence)
        pur(purpose or objective)
        rsn(reason)
        seq(sequence)
  qualification relation
        bas(basis for expressing a standard)
        cnt(content, namely)
        man(manner)
        mod(modification)
        nam(name)
        per(proportion, rate or distribution)
        pof(part-of)
        pos(possessor)
        qua(quantity)
```

Definitions of Relations

The following are the relations defined according to the above principles. A relation label is represented as strings of 3 characters or less

(1) agt (agent)

Agt defines a thing (an agent) in focus that initiates an action. It can be a relation between:

```
do--agt->thing
action--agt->thing
```

An agent initiate an action indicated by "do" or "action" and is thought of as having a direct role in making the action happen.

EXAMPLES AND READINGS

break--agt->John	John breaks ...
translate--agt->computer	computer translates ...
run--agt->car	car runs ...
destroy--explosion	explosion destroys ...

(2) and (conjunction)

And defines a partner to have conjunctive relation. It can be a relation between:

```
and(concept, concept)
```

A conjunction is defined as the relation between a concept, and another concept.

EXAMPLES AND READINGS

quickly--and->easily	... easily and quickly
dance--and->sing	... singing and dancing
Mary--and->John	... John and Mary

(3) aoj (thing with attribute)

Aoj defines a thing that is in a state or has an attribute. It can be a relation between:

```
be--aoj->thing
thing--aoj->thing
```

EXAMPLES AND READINGS

red--aoj->leaf	… leaf is red.
available--aoj->information	This information is available for …
nice--aoj->ski	Skiiing is nice.
teacher-->aoj->John	John is a teacher.
I<-aoj--have--obj->pen	I have a pen.
know--aoj->John	John knows …

(4) bas (standard (basis) of comparison)

Bas defines a thing used as the basis (standard) of comparison. It can be a relation between:

```
be--bas->thing
do--bas-> thing
how--bas->thing
```

EXAMPLES AND READINGS

more--bas->7	more than seven.
more--bas->Jack	Betty weighs more than Jack (does).
beautiful--man->more--bas->rose	A tulip is more beautiful than a rose
(quiet(aoj>thing)--man->more--bas->shy)--aoj->John	John is more quiet than shy.
prefer--bas->(liveing--plc->city)	Many people prefer living in the country to living in a city

(5)ben (beneficiary)

Ben defines an indirectly related beneficiary or victim of an event or state. It can be a relation between:

```
predicative concept--ben->thing
```

EXAMPLES AND READINGS

give-->ben->country	To give one's life for one's country.
good--ben->John	It is good for John to …

(6) cag (co-agent)

Cag defines a thing not in focus that initiates an implicit event that is done in parallel. It can be a relation between:

```
do--cag->thing
```

EXAMPLES AND READINGS

walk--cag->John	To walk with John
live--cag->aunt	To live with … aunt

(7) cao (co-thing with attribute)

Cao defines a thing not in focus that is in a parallel state. It can be a relation between:

```
be--cao->thing
thing--cao->thing
```

EXAMPLES AND READINGS

exist--cao->you	… be with you

(8) cnt (content)

Cnt defines or show the content of a concept. It can be a relation between:

```
concept--cnt->concept
```

EXAMPLES AND READINGS

Internet--cnt->amalgamation	The Internet: an amalgamation
language generator--cnt->"deconverter"	a language generator "deconverter"…
risk--cnt->(losing--obj->money	the risk of losing money

(9) cob (affected co-thing)

Cob defines a thing that is directly affected by an implicit event done in parallel or an implicit state in parallel.

```
predicative concept--cob->thing
```

EXAMPLES AND READINGS

die--cob->Mary	… dead with Mary
John<-obj--njure--cob->friend.pl--pos->he	John was injured in the accident with his friends

(10) con (condition)

Con defines a non-focused event or state that conditions a focused event or state.

```
predicative concept--con->concept
```

EXAMPLES AND READINGS

you<-aoj--tired--con->go	If you are tired, we will go straight home.

(11) coo (co-occurrence)

Coo defines a co-occurrent event or state for a focused event or state.

```
predicative concept--coo->predicative concept
```

EXAMPLES AND READINGS

cry--coo->run	… was crying while running
red--coo->hot	… is red while … is hot

(12) dur (duration)

Dur defines a period of time during which an event occurs or a state exists.

```
predicative concept--dur->predicative concept
predicative concept--dur->event
predicative concept--dur->period
predicative concept--dur->state
```

EXAMPLES AND READINGS

work--work->hour--qua->9	… work nine hours (a day)
talk--dur->meeting	… talk … during meeting
come--dur->absence	… come during (my) absence

(13) equ (equivalent)

Equ defines an equivalent concept.

```
concept--equ->concept
```

EXAMPLES AND READINGS

deconverter--equ->generator	the deconverter (a language generator)

(14) fmt (range: from-to)

Fmt defines a range between two things.

```
thing--fmt->thing
```

EXAMPLES AND READINGS

z--fmt->a	the alphabets from a to z
New York--fmt->Osaka	the distance from Osaka to New York
Friday--fmt->Monday	the weekdays from Monday to Friday

(15) frm (origin)

Frm defines an initial state of a thing or a thing initially associated with the focused thing.

```
thing--frm->thing
thing->frm->be
```

EXAMPLES AND READINGS

visitor--frm->Japan	a visitor from Japan

(16) gol (goal: final state)

Gol defines a final state of object or a thing finally associated with the object of an event.

```
be--gol->thing
do--gol->thing
do--gol->be
occur--gol->thing
occur--gol->be
```

EXAMPLES AND READINGS

change--gol->red	the lights changed from green to red
deposit--gol->account	millions were deposited in a Swiss bank account

(17) icl (included/a kind of)

Icl defines an upper concept or a more general concept.

```
concept--icl->concept
```

EXAMPLES AND READINGS

mammal--icl->animal	a mammal is a (kind of) animal

(18) ins (instrument)

Ins defines an instrument to carry out an event.

```
do--ins->concrete thing
```

EXAMPLES AND READINGS

look--ins->telescope	look at stars through [with] a telescope
write--ins->pencil	write [draw] with a pencil
cut--ins->scissors	He cut the string with a pair of scissors

(19) int(intersection)

Int defines an intersection between concepts. a partner to take an intersection

```
concept--int->concept
```

(20) iof (an instance of)

Iof defines a class concept that an instance belongs to.

```
concept--iof->concept
```

EXAMPLES AND READINGS

Tokyo--iof->city in Japan	Tokyo is a city in Japan

(21) man (manner)

Man defines a way to carry out an event or the characteristics of a state.

```
predicative concept--man->how
```

EXAMPLES AND READINGS

move--man->quickly	move quickly
visit--man->often	I often visit him.
beautiful--man->very	it is very beautiful.

(22) met (method or means)

Met defines a means to carry out an event.

```
do--met->abstract thing
do--met->do
```

EXAMPLES AND READINGS

solve--met->dynamics	... solve ... with dynamics
solve--met->algorithm	... solve ... using ... algorithm
separate--met->cut	... separate ... by cutting ...

(23) mod (modification)

Mod defines a thing that restricts a focused thing.

```
thing--mod->thing
thing--mod->attributive concept
```

EXAMPLES AND READINGS

story--mod->whole	the whole story
plan--mod->master	a master plan
part--mod->main	the main part

(24) nam (name)

Nam defines a name of a thing.

```
thing--nam->thing
```

EXAMPLES AND READINGS

tower--nam->Tokyo	Tokyo tower

(25) obj (affected thing)

Obj defines a thing in focus that is directly affected by an event or state.

```
predicative concept--obj->thing
```

EXAMPLES AND READINGS

move--obj>table	the table moved.
melt--obj->sugar	the sugar melts into ...
cure--obj->patient	to cure the patient.
have--obj->pen	I have a pen.

(26) opl (object place)

Opl defines a place in focus affected by an event.

```
do--opl->thing
occur--opl->thing
```

EXAMPLES AND READINGS

pat--opl->shoulder	... pat ... on shoulder
cut--opl->middle	... cut ... in middle

(27) or (disjunction)

Or defines a disjunctive relation between two concepts.

```
concept--or->concept
```

EXAMPLES AND READINGS

leave--or->stay	Will you stay or leave?
blue--or->red	Is it red or blue?
Jack--or->John	Who is going to do it, John or Jack?

(28) per (proportion, rate or distribution)

Per defines a basis or unit of proportion, rate or distribution.

```
thing--per->thing
```

EXAMPLES AND READINGS

8<-qua--hour--per->day	eitgh hours a day
2<-qua--time--per->week	... twice a week

(29) plc (place)

Plc defines a place where an event occurs, or a state that is true, or a thing that exists.

```
predicative concept--plc->place
thing--plc->place
```

EXAMPLES AND READINGS

cook--plc->kitchen	... cook ... in the kitchen
sit--plc->beside	... sit beside me
cool--plc->here	It's cool here.

(30) plf (initial place)

Plf defines a place where an event begins or a state that becomes true.

```
predicative concept--plf->thing
```

EXAMPLES AND READINGS

travel--plf->Tokyo	travelling from Tokyo
deep--plf->there	The sea is deep from there to here.

(31)plt (final place)

Plt defines a place where an event ends or a state that becomes false.

```
predicative concept--plt->thing
```

EXAMPLES AND READINGS

travel--plt->Boston	to travel to Boston
deep--plt->here	The sea is deep from there to here

(32) pof (part-of)

Pof defines a concept of which a focused thing is a part.

```
thing--pof->thing
```

EXAMPLES AND READINGS

preamble--pof->document	the preamble of a document
initial--pof--machine translation	the initials of Machine Translation

(33) pos (possessor)

Pos defines the possessor of a thing.

```
thing--pos->volitional thing
```

EXAMPLES AND READINGS

dog--pos->John	John's dog
book--pos->I	my book

(34) ptn (partner)

Ptn defines an indispensable non-focused initiator of an action

```
do--ptn->thing
```

EXAMPLES AND READINGS

compete--ptn->John	... compete with John
share--ptn->poor	... share ... with the poor
collaborate--ptn->he	... collaborate with him ...

(35) pur (purpose or objective)

Pur defines the purpose or objective of an agent of an event or a purpose of a thing that exists.

```
do--pur->do
do--pur->thing
thing--pur->concept
```

EXAMPLES AND READINGS

come--pur->see	... come to see you
work--pur->money	... work for money
budget--pur->research	our budget for research

(36) qua (quantity)

Qua defines the quantity of a thing or unit.

```
thing--qua->quantity
```

EXAMPLES AND READINGS

coffee--qua->cup--qua->cup--qua->2	Two cups of coffee
kilogram--qua->many	many kilograms
dog--qua->2	two dogs

(37) rsn (reason)

Rsn defines a reason why an event or a state happens.

```
predicative concept--rsn->predicative concept
predicative concept--rsn->thing
```

EXAMPLES AND READINGS

go.not--rsn->rain	... didn't go because of the rain
start--rsn->(Mary<-agt--arrive)	They can start because Mary arrived.
city<-aoj--known--rsn->beauty	a city known for its beauty

(38) scn (scene)

Scn defines a scene where an event occurs, or state is true, or a thing exists.

```
predicative concept--scn->thing
thing--scn->thing
```

EXAMPLES AND READINGS

win--scn->contest	... win a prize in a contest
appear--scn->program	... appear on a TV program
play--scn->movie	... play in movie

(39) seq (sequence)

Seq defines a prior event or state of a focused event or state.

```
predicative concept--seq->predicative concept
```

EXAMPLES AND READINGS

leap--seq->look	Look before you leap.
red--seq->green	It was green and then red.
take off-->seq->come in	She came in and took her coat off.

(40) src (source: initial state)

Src defines the initial state of an object or thing initially associated with the object of an event.

```
do--src->thing
do--src->be
occur--src->thing
occur--src->be
```

EXAMPLES AND READINGS

change--src->red	The lights changed from green to red.
withdraw--src->stove	I quickly withdrew my hand from the stove.

(41) tim (time)

Tim defines the time an event occurs or a state is true.

```
predicative concept--tim->time
thing--tim->time
```

EXAMPLES AND READINGS

leave--tim->Tuesday	... leave on Tuesday
do--tim->o'clock	... do ... at ... o'clock
start--tim->come	Let's start when ... come

(42) tmf (initial time)

Tmf defines the time an event starts or a state becomes true.

```
predicative concept--tmf->time
thing--tmf->time
```

EXAMPLES AND READINGS

work--tmf->morning	... work from morning to [till] night
change--tmf->live	... has changed ... since I have lived here.

(43)tmt (final time)

Tmt defines a time an event ends or a state becomes false.

```
predicative concept--tmt->time
thing--tmt->time
```

EXAMPLES AND READINGS

work--tmt->night	... work from moning to [till] night
full--tmt->tomorrow	... be full till tomorrow

(44) to (destination)

To defines a final state of a thing or a final thing (destination) associated with the focused thing.

```
thing--to->thing
```

EXAMPLES AND READINGS

train--to->London	a train for London
letter--to--you	a letter to you

(45) via (intermediate place or state)

Via defines an intermediate place or state of an event.

```
do--via->thing
occur--via->thing
```

EXAMPLES AND READINGS

go--via->New York	... go ... via New York
bike--via->Alps	... bike ... through the Alps
drive--via->tunnel	... drive ... by way of the tunnel

2.1.3 Attributes

Attributes are mainly for the purpose to describe the subjectivity information of sentences. We adopt the attributes set of the UNL. They show what is said from the speaker's point of view: how the speaker views what is said. This includes phenomena technically called "speech acts", "propositional attitudes", "truth values", etc. Attributes are also used to express the range of concepts such as the concept indicate generic type of concept and so forth. This time, we newly introduce attributes to express logical expressions in order to strengthen the expressibility of the CWL.

Relations and concepts are used to describe the objectivity information of sentences. Attributes modify concepts including hyper node (compound concepts) to indicate subjectivity information such as about how the speaker views these states-of-affairs and his attitudes toward them and to indicate the property of the concepts.

Attributes are divided into the following groups:

- Describing logic characters / properties of concepts
- Times with respect to the speaker
- Speaker's view on aspects of event
- Speaker's view of reference to concepts
- Speaker's view of emphasis, focus and topic
- Speaker's attitudes
- Speaker's feelings and judgments

```
Attribute:
    attribute of concept
        attribute of nominal concept
        attribute of predicative concept
            aspect (view on aspects of event)
                begin           :beginning of an event or a state
                complete        :finishing/completion of a (whole) event
                continue        :continuation of an event
                custom          :customary or repetitious action
                end             :end/termination of an event or a state
                experience      :experience
                progress        :an event is in progress
                repeat          :repetition of an event
                state           :final state or the existence of the
                                 object on which an action has been taken
            time (with respect to speaker)
                past            :happened in the past
                present         :happening at present
                future          :will happen in future
            view of emphasis, focus and topic
                contrast        :contrasted concept
                emphasis        :emphasized concept
                entry           :entry or main UW of a sentence or a scope
                qfocus          :focused concept of a question
                theme           :instantiates an object from a different class
                title           :title
                topic           :topic
            attitudes(modality)
                affirmative     :affirmation
                confirmation    :confirmation
                exclamation     :exclamation
                humility        :in a humility manner
                imperative      :imperative
                interrogative   :interrogation
                invitation      :inducement
                polite          :polite way
                request         :request
                respect         :respectful way
                vocative        :vocative
            feelings and judgments
```

```
    ability        :ability, capability of doing something
    get-benefit    :speaker's feeling of receiving
                    benefits through the fact or result of
                    something (to be) done by somebody else
    give-benefit   :speaker's feeling of giving benefits
                    by doing something for somebody else
    conclusion     :logical conclusion due to a certain condition
    consequence    :logical consequence
    sufficient     :sufficient condition
    consent        :consent feeling of the speaker about something
    dissent        :dissent feeling of the speaker about something
    grant          :to give/get consent/permission to do something
    grant-not      :not to give consent to do something
    although       :something follows against [contrary
                    to] or beyond expectation
    discontented   :discontented feeling of the speaker
                    about something
    expectation    :expectation of something
    wish           :wishful feeling, to wish something is
                    true or has happened
    insistence     :strong determination to do something
    intention      :intention about something or to do something
    want           :desire to do something
    will           :determination to do something
    need           :necessity to do something
    obligation     :obligation to do something according
                    to (quasi-) law, contract, or …
    obligation-not :obligation not to do something, forbid
                    to do something according to (quasi-)
                    law, contract or …
    should         :to do something as a matter of course
    unavoidable    :unavoidable feeling of the speaker
                    about doing something
    certain        :certainty that something is true or happens
    inevitable     :logical inevitability that something
                    is true or happens
    may            :practical possibility that something
                    is true or happens
    possible       :logical possibility that something is
                    true or happens
    probable       :(practical) probability that something
                    is true or happens
    rare           :rare logical possibility that
                    something is true or happens
    unreal         :unreality that something is true or happens
    admire         :admiring feeling about something
    blame          :blameful feeling about something
    contempt       :contemptuous feeling about something
    regret         :regretful feeling about something
    surprised      :surprised feeling about something
    troublesome    :troublesome feeling about occurrence
                    of something
logicality
    transitive     :has transitivity
    symmetric      :has symmetricity
    identifiable   :can identify the subject
    disjointed     :all element concept have no common
                    instance all connected concept do not
                    share instances
view of reference
    generic        :generic concept
    def            :already referred
    indef          :non-specific class
    not            :complement set
    ordinal        :ordinal number
modifying attribute on aspect
    just           :expresses an event or a state that has
                    just begun or ended/completed
    soon           :expresses an event or a state that is
                    about to begin or end/completed
    yet            :expresses an event or a state that has
                    not yet started or ended/completed,
                    together with not
```

2.1.3.1 Time with respect to the Speaker

Where does the speaker situate his description in time, taking his moment of speaking as a point of reference? A time before he spoke? After? At approximately the same time? This is the information that defines "narrative time" as past, present or future. These Attributes are attached to the main predicate.

Although in many languages this information is signaled by tense markings on verbs, the concept is not tense, but "time with respect to the speaker:. The clearest example is the simple present tense in English, which is not interpreted as the present time, but as "independently of specific times".

Consider the example:

> The earth is round.

This sentence is true in the past, present and future, independently of the speaker's time, so although the tense is "present" it is not interpreted as the present time.

past	happened in the past	ex) It was snowing yesterday
present	happening at present	ex) It is raining hard.
future	will happen in future	ex) He will arrive tomorrow

2.1.3.2 The Speaker's View of Aspect

A speaker can emphasize or focus on part of an event or treat it as a whole unit. This is closely linked to how the speaker places the event in time. These Attributes are attached to the main predicate.

The speaker can focus on the beginning "begin" of the event, looking forward to it "begin.soon", or backward to it "begin.just".

He can also focus on the end "end" or completion "complete" of the event, looking forward to it "end.soon" or "complete.soon", or backward to it "end.just" or "complete.just".

He can focus on the middle "progress" or continuation "continue" of the event.

The speaker can choose to focus on the lasting effects or final state of the event "state" or on the event as a repeating unit "repeat", experience "experience" or custom "custom".

He can also focus on the incompleteness or the fact that it has not yet happened, by using "yet".

begin	beginning of an event or a state	ex) It began to work again.
complete	finishing/completion of a (whole) event.	ex) I've looked through the script. look.entry.complete
continue	continuation of an event	ex) He went on talking. talk.continue.past
custom	customary or repetitious action	ex) I used to visit [I would often go] there when I was a boy. visit.end.present
end	end/termination of an event or a state	ex) I have done it. do.end.present
experience	experience	ex) Have you ever visited Japan? visit.experience.interrogation ex) I have been there. visit.exterience
progress	an event is in progress	ex) I am working now. work.progress.present
repeat	repetition of an event	ex) It is so windy that the tree branches are knocking against the roof. knock.entry.present.repeat
state	final state or the existence of the object on which an action has been taken	ex) It is broken. break.state

These attributes are used to modify the attributes above, to express a variety of aspects of natural languages.

just	Expresses an event or a state that has just begun or ended/completed	ex) He has just come. come.complete.just
soon	Expresses an event or a state that is about to begin or end/completed	ex) The train is about to leave. leave.begin.soon
yet	Expresses an event or a state that has not yet started or ended/completed, together with "not".	ex) I have not yet done it. do.complete.not.yet

2.1.3.3 Reference to the range of a concept

Whether an expression refers to a single individual, a small group or a whole set is often not clear. The expression "the lion" is not sufficiently explicit for us to know whether the speaker means "one particular lion" or "all lions". Consider the following examples:

> The lion is a feline mammal.
> The lion is eating an antelope.

In the first example, it seems reasonable to suppose that the speaker understood "the lion" as "all lions", whereas in the second example as "one particular lion".

The following Attributes are used to make explicit what the speaker's view of reference seems to be.

generic	generic concept	ex) The dog is a faithful animal.
def	already referred	ex) the book you lost
indef	non-specific class	ex) There is a book on the desk.
not	complement set	ex) Don't be late!
ordinal	ordinal number	ex) the 2nd door

These attributes are usually attached to UWs that denote things.

2.1.3.4 The Speaker's View of Emphasis, Focus and Topic

The speaker can choose to focus or emphasize parts of a sentence to show how important he thinks they are in the situation described. This is often related to sentence structure.

		For instance, "but" in the examples below is used to introduce a word or phrase that contrasts with what

contrast	Contrasted UW	was said before. ex) It wasn't the red one but the blue one. ex) He's poor but happy.
emphasis	Emphasized UW	ex) I do like it.
entry	Entry or main UW of a sentence or a scope	ex) He promised (entry of the sentence) that he would come (entry of the scope)
qfocus	Focused UW of a question	ex) Are you painting the bathroom blue? To this question, the answer will be `No, I'm painting the LIVING-ROOM blue'
theme	Instantiates an object from a different class	
title	Title	
topic	Topic	ex) He("topic") was killed by her. ex) The girl("topic") was given a doll. ex) This doll("topic") was given to the girl.

One concept marked with "entry" is essential for each CWL expression or in a compound concept.

2.1.3.5 The Speaker's Attitudes

The speaker can also express, directly or indirectly, what his attitudes or emotions are towards what is being said or whom it is being said to. This includes respect and politeness towards the listener and surprise toward what is being said.

affirmative	**Affirmation**
confirmation	**Confirmation** ex) You won't say that, will you? ex) It's red, isn't it? ex) Then you won't come, right?
exclamation	**Feeling of exclamation** ex) kirei na! (`How beautiful (it is)!' in Japanese) ex) Oh, look out!
humility	**In a humility manner to express something** ex) That is quite impossible for the likes of me.@humility.
imperative	**Imperative** ex) Get up! ex) You will please leave the room.
interrogative	**Interrogation** ex) Who is it?
invitation	**Inducement to do something** ex) Will / Won't you have some tea? ex) Let's go, shall we?
polite	**Polite way to express something** ex) Could you (please)... ex) If you could ... I would ...
request	**Request** ex) Please don't forget...
respect	**Respectful feeling. In many cases, some special words are used.** ex) o taku (`(your) house' in Japanese) ex) Good morning, sir.
vocative	**Vocative** ex) Boys, be ambitious!

2.1.3.6 The Speaker's Feelings, Judgement and Viewpoint

These attributes express the speaker's feelings or how the speaker views or judges what is said.

This sort of subjective information is very much dependent on the type of language. It should be possible to express every kind of subjective information from all languages. Thus, the development of the attributes is open to the developers of each language, who can introduce a new attribute when no current attribute expresses its meaning. The new attribute must be also introduced in the same way.

The following attributes are used to clarify the speaker's viewpoint information.

- Ability

ability	**Ability, capability of doing something** ex) The child can 't walk yet. ex) He can speak English but he can't write it very well.

- Beneficialty

get-benefit	**Speaker's feeling of receiving benefits through the fact or result of something (to be) done by somebody else** ex) I'll have my secretary type the letter. *In Japanese the auxiliary verb of `~te morau' is used to express the getting benefits feeling. For instance it is frequently used in a sentence in the sense of `to have somebody do something' in Japanese.
give-benefit	**Speaker's feeling of giving benefits by doing something for somebody else** ex) Be kind to old people. *In Japanese the auxiliary verb of `~te ageru' is used to express the giving benefits feeling. For instance

	it is frequently used in a sentence in the sense of `Be kind to old people' in Japanese.

- Conclusion

conclusion	**Logical conclusion due to a certain condition** ex) He is her husband; she is his wife.
consequence	**Logical consequence** ex) He was angry, wherefore I left him alone.

- Condition

sufficient	**Sufficient condition** ex) only have to

- Consent and dissent

consent	**Consent feeling of the speaker about something**
dissent	**Dissent feeling of the speaker about something** ex) But that's not true.
grant	**To give/get consent/permission to do something** ex) Can I smoke in here? ex) You may borrow my car if you like.
grant-not	**Not to give consent to do something** ex) You {mustn't/are not allowed to/may not} borrow my car.

- Expectation

although	**Something follows against [contrary to] or beyond expectation** ex) Although he didn't speak, I felt a certain warmth in his manner.
discontented	**Discontented feeling of the speaker about something** ex) (I'll tip you 10 pence.) But that's not enough!
expectation	**Expectation of something** ex) Children ought to be able to read by the age of 7. ex) If you leave now, you should get there by five o'clock.
wish	**Wishful feeling, to wish something is true or has happened** ex) If only I could remember his name! (~I do wish I could remember his name!) ex) You might have just let me know.

- Intention

insistence	**Strong determination to do something** ex) He will do it, whatever you say.
intention	**Intention about something or to do something** ex) He shall get this money. (Speaker's intention) ex) We shall let you know our decision.
want	**Desire to do something** ex) I want to go France.
will	**Determination to do something** ex) I'll write as soon as I can. ex) We won't stay longer than two hours.

- Necessity and obligation

need	**Necessity to do something** ex) You need to finish this work today. ex) I must be going now. ex) I always have to work hard.
obligation	**Obligation to do something according to (quasi-) law, contract, or ...** ex) The vendor shall maintain the equipment in good repair. ex) You must come by nine.
obligation-not	**Obligation not to do something, forbid to do something according to (quasi-) law, contract or ...** ex) Cars must not park in front of the entrance. ex) No smoking
should	**To do something as a matter of course** ex) You should do as he says. ex) You ought to start at once.
unavoidable	**Unavoidable feeling of the speaker about doing something** ex) I could not help speaking the truth.

- Possibility

certain	**Certainty that something is true or happens** ex) If Peter had the money, he would have bought a car. ex) They should be home by now.
inevitable	**Logical inevitability that something is true or happens** ex) All living things must die.
may	**Practical possibility that something is true or happens** ex) It may be true. ex) It could be.
possible	**Logical possibility that something is true or happens** ex) Anybody can make mistakes. ex) If Peter had the money, he would buy a car.
probable	**(Practical) probability that something is true or happens** ex) That would be his mother. ex) He must be lying.
rare	**Rare logical possibility that something is true or happens** ex) If such a thing should happen, what shall we do? ex) If I should fail, I will [would] try again.
unreal	**Unreality that something is true or happens** ex) If we had enough money, we could buy a car. ex) If Peter had the money, he could buy a car.

- Emotion

admire	**Admiring feeling of the speaker about something**
blame	**Blameful feeling of the speaker about something** ex) A sailor, and afraid of the sea!
contempt	**Contemptuous feeling of the speaker about something** ex) You never could do it *In Japanese the postpositional particles of `nado', `nanka' or `nante' as in `kimi nado niha..' can be used to express the contemptuous feeling of the speaker about the target, mainly in a negative sentence
regret	**Regretful feeling of the speaker about something** ex) It's a pity that he should miss such a golden opportunity.
surprised	**Surprised feeling of the speaker about something** ex) (He has succeeded!) But that's great!
troublesome	**Troublesome feeling of the speaker about the occurrence of something** ex) My house was [I had my house] broken into.@troublesome yesterday. *There is a troublesome feeling of the speaker when using a passive form of the verb in this case in Japanese.

2.1.3.7 Convention

Typical CWL structures can be expressed by attributes to avoid the complexity of enconverting and deconverting. What marks are used for enclosing a word or phrase can also be expressed by attributes. The attributes for indicating enclosure must be attached to the scope node of the enclosed phrase if it consists of a (set of) binary relation(s) of CWL.

passive	passive form	ex) Being bitten.@passive by a dog ...
pl	more than one	ex) children: child(icl>young person).@pl
angle_bracket	< > are used	
brace	{ } are used	
double_parenthesis	(( )) are used	
double_quote	" " are used	
parenthesis	( ) are used	
single_quote	' ' are used	
square_bracket	[ ] are used	

2.2 CWL platform

As we do not have any system for CWL at this moment, we need to use the UNL system for input CWL and output what is expressed in CWL in natural languages. We basically use the UNL system as a platform of CWL.

The CWL platform consists of CWL Editor, CWL converter and the UNL System consisting of Enconverter (which convert natural languages into UNL), Deconverter (which convert UNL into natural languages).

Since compatibility of the three types of expression of CWL.unl, CWL.cdl, CWL.rdf, The UNL system together with conversion system allow people to make web pages in CWL.unl, CWL.cdl, CWL.rdf and also allow people to see those web pages in their mother tongues.

Beside the UNL system, we provided conversion system (CWL converter) among CWL.unl, CWL.cdl and CWL.rdf. And also we provide necessary vocabulary for CWL based on the UNLKB which provides semantic background of Universal Words of UNL. Knowledge on words (CWL ontology) of each language is stored in UNLKB (CWL.unl), CDD.nl (CWL.cdl) and OWL (CWL.rdf).

3. CWL.unl

The UNL is an acronym for `Universal Networking Language'. The CWL.unl is a prepsentation of CWL in UNL. The UNL including UNLKB and UWs was developed under United Nations University / Institutes of Advanced Sturdy in 1996, and research and development were transferred to the UNDL Foundation in 2001.(http://www.undl.org/unlsys/uw/unlkb.htm)

3.1 UNL

The UNL expresses information or knowledge in the form of semantic network with hyper-node. UNL semantic network is made up of a set of binary relations, each binary relation is composed of a relation and two UWs that hold the relation. A binary relation of UNL is expressed in the following format:

```
<relation> (  <uw1>, <uw2> )
```

In <relation>, one of the relations defined in the UNL Specifications is described. In <uw1> and <uw2>, the two UWs that have the relation given by <relation> are described. Semantic network of UNL expression is a directed graph by means of the binary relations with direction. The three elements of each binary relation have the following interrelationship:

```
<uw1> - <relation> → <uw2>
```

Such a binary relation is interpreted as that:

> the UW given in <uw2> plays the role indicated by the relation given in <relation> held by the UW given in <uw1>; whereas the UW given in <uw1> holds the relation given in <relation> with the UW given in <uw2>.

A UNL expression is a hyper semantic network. That is, each node of the graph, <uw1> and <uw2> of a binary relation, can be replaced with a semantic network. Such a node consists of a semantic network of a UNL expression and is called a `scope'. A scope can be connected with other UWs or scopes. The UNL expressions in a scope is distinguished from others by assigning an ID to the <relations> of the set of binary relations that belong to the scope.

The general description format of binary relations for a hyper-node of UNL expression is the following:

```
<relation> : <scope-id> ( <node1>, <node2> )
```

Where,

- <scope-id> is the ID for distinguishing a scope.
- <scope-id> is not necessary to specify when a binary relation does not belong to any scope.
- <node1> and <node2> can be a UW or a <scope node>.
- A <scope node> is given in the format of `:<scope-id>'.

3.1.1 UNL Document

UNL expressions are provided in the format of UNL documents. A UNL document is a text file that includes the original sentences, UNL document tags, UNL expressions and etc.

A UNL document is enclosed with tags `[D:<dinf>]' and `[/D]'. Within these tags, each paragraph is enclosed with a pair of tags `[P:<p_num>]' and `[/P]', and each sentence is enclosed with a pair of tags `[S:<s_num>]' and `[/S]'. Inside a sentence, the source text is enclosed with `{org:<l_tag>}' and `{/org}', its UNL expression is enclosed with `{unl:<uinf>}' and `{/unl}'. Sentences of target languages can also be stored in the UNL document. Each target sentence is enclosed with a pair of language tags `{<l_tag>}' and `{</l_tag>}' following the UNL expression of each sentence.

Description format of a UNL document is the following:

<UNL Document>	::= "[D:" <dinf> "]" {"[P:" <paragraph number> "]" {"[S:" <sentence number> "]" <sentence> "[/S]"} ... "[/P]" } ... "[/D]"
<dinf>	::= <document name> "," <author name> ["," <document ID> "," <date> "," <email address>]
<document name>	::= "dn=" <character string>
<author name>	::= "on=" <character string>
<document ID>	::= "did=" <character string>
<date>	::= "dt=" <character string>
<email address>	::= "mid=" <character string>
<sentence>	::= "{org:" <l-tag> ["=" <code>] "}" <source sentence> "{/org}" "{unl" [":" <uinf>] "}" <UNL expression> "{/unl}" { "{" <l-tag> ["=" <code>] [":" <sinf> "]" "}" <target sentence> "{/" <l-tag> "}" } ... /* whole information that necessary for a sentence */
<l-tag>	::= "ab" | "cn" | "de" | "el" | "es" | "fr" | "id" | "hd" | "it" | "jp" | "lv" | "mg" | "pg" | "ru" | "sh" | "th"; /* language codes : language tags */
<code>	::= <character code name>
<character code name>	::= <character string>
<source snetence>	::= <character string>
<target sentence>	::= <character string>
<uinf>	::= <system name> "," <post-editor name> "," <reliability> ["," <date> "," <email address>]
<sinf>	::= <system name> "," <post-editor name> "," <reliability> ["," <date "," <email address>]
<system name>	::= "sn=" <character string>
<post-editor name>	::= "pn=" <character string>
<reliability>	::= "rel=" <a number>
<paragraph number>	::= <a number>
<sentence number>	::= <a number>

The tags used in a UNL document are the following:

[D:<dinf>]	indicates the Beginning of a document and the necessary information about the document
[/D]	indicates the End of a document
[P:<p_num>]	indicates the Beginning of a paragraph
[/P]	indicates the End of a paragraph
[S:<s_num>]	indicates the Beginning of a sentence and the sentence number

[/S]	indicates the End of a sentence
{org:<l_tag>=<code>}	indicates the Beginning of an original/source sentence, language and character code, "=<code>" can be omitted.
{/org}	indicates the End of an original sentence
{unl:<uinf>}	indicates the Beginning of the UNL expressions of a sentence and necessary information, `:<uinf>' can be omitted.
{/unl}	indicates the End of the UNL expressions of a sentence
{<l_tag>}	indicates the Beginning of a target sentence of the language indicated by <l_tag>
{/<l_tag>}	indicates the End of a target sentence of the language indicated by <l_tag>

See the following section about <UNL expression>.

3.1.2 UNL Expression

A UNL expression of a sentence is identified with the following tags: {unl} and {/unl}.

Any component, such as a word, phrase and, of course, a sentence of a natural language can be represented with UNL expressions. A UNL expression therefore consists of a UW or a (set of) binary relation(s). In UNL documents, a UNL expression for a sentence is enclosed by the tags {unl} and {/unl} inside [S] and [/S]. If a UNL expression consists of a UW, this UW should be enclosed further by the tags [W] and [/W]. If necessary, the whole sentence can also be expressed as a scope. In this case, the Compound UW-ID of the scope should be enclosed by [W] and [/W].

There are two forms for expressing UNL expressions, one is the table form and the other is the list form. The table form is made up of a set of binary relations, and each binary relation is expressed by connecting the two related UWs directly. And the list form is divided into two parts: a list of UWs corresponding IDs and a list of binary relations described by the IDs. The table form of a UNL expression is more readable than the list form, but the list form of a UNL expression is more compact than the table form. These two forms are convertible with each other.

3.1.2.1 The table form of UNL expression

- A UNL expression consists of a set of binary relations:

```
{unl}
<binary relation>
...
{/unl}
```

- A UNL expression consists of a UW:

```
{unl}
[W]
<UW><attribute list>
[/W]
{/unl}
```

- A UNL expression consists of a scope:

```
{unl}
[W]
n:m <compound UW-ID><attribute list>
[/W]
<binary relation>
...
{/unl}
```

Each tag and binary relation should end with a return code: `0x0a'.

SYNTAX OF BINARY RELATION

Description format of a binary relation of the table form is the following:

<binary relation>	::= <relation> [":"<compound UW-ID>] "(" {{ <UW_1> [":" <UW-ID_1>]} | { ":" <compound UW-ID_1> }}[<attribute list>] "," {{<UW_2> [":" <UW-ID_2>]} | { ":" <compound UW-ID_2> }}[<attribute list>] ")"
<relation>	::= a relation label, defined in `Chapter 2 Relations'
<UW>	::= a UW, see `Chapter 3 Universal Words'
<attribute list>	::= { `.' <attribute>} ...
<attribute>	::= an attribute, see `Chapter 4 Attributes'
<UW-ID>	::= two alphanumeric characters of '0' - '9' and 'A' - 'Z'
<compound UW-ID>	::= two digits of `00' - `99'. `00' must be used for the main sentence and can be ommited.

COMPOUND UW-ID

A UNL expression can include more than one scope. Compound UW-IDs are for identifying each concept specified by compound UWs (scopes) in a UNL expression. A scope is a group of binary relations that can be referred to as a UW by indicating its compound UW-ID in the format of `:<Compound UW-ID>'. A node described in this way in the UNL expression network that refers to a scope is called a `Scope Node'.

For details about the scope please refer to `3.2 Compound UWs'.

UW-ID

UW-IDs are for identifying each concept specified by UWs in a UNL expression. If a UW appears in a UNL expression more than once and means different concepts (things or events), a unique UW-ID must be given to each concept of the UWs.

The following shows an example of UNL expressions of the sentence `I can hear a dog barking outside':

```
{unl}
agt(hear(icl>perceive(agt>person,obj>thing)).@entry,    I)
obj(hear(icl>perceive(agt>person,obj>thing)).@entry,    :01)
agt:01(bark(agt>dog).@entry,    dog(icl>canine))
plc:01(bark(agt>dog).@entry,    outside(icl>place))
{/unl}
```

In above UNL expression, `**agt**', `**obj**' and `**plc**' are relation labels, `**I**', `**bark(agt>dog)**', `**dog(icl>canine)**', `**hear(icl>perceive(agt>person,obj>thing))**' and `**outside(icl>place)**' are UWs. `a dog barking outside' is expressed by a scope, and `**01**' is given as the compound UW-ID to the scope. `:01' appears in the position of a UW is the scope node to refer to the scope. Binary relations indicated by the Compound UW-ID define the contents of the scope.

3.1.2.2 The list form of UNL expression

The list form of a UNL expression consists of a set of UWs and a set of encoded binary relations (expressed by UW-IDs) of a sentence. In case a whole sentence is treated as a scope, the Compound UW-ID of the scope for the sentence can be included in the UW list between [W] and [/W].

```
{unl}
[W]
{<UW> | {`:'<compound UW-ID>}}[<attribute list>]":" <UW-ID>
…
[/W]
[R]
<binary relation by UW-IDs>
…
[/R]
{/unl}
```

The tags used above have the following meanings.

[W]	indicates the Beginning of the UW list.
[/W]	indicates the End of the UW list.
[R]	indicate the Beginning of the encoded binary relations.
[/R]	indicates the End of the encoded binary relations.

Each tag, encoded binary relation and UW should end with a return code: `0x0a'.

UW List

UWs of a UNL expression must be listed between [W] and [/W] with different (unique) UW-IDs for different concepts. This means that the same UW expression but expressing different concepts (instances) must be given different UW-IDs. A scope must be defined again in the UW list.

Syntax of an encoded binary relation

<binary relation by UW-IDs>	:= <UW-ID$_1$><relation>[`:'<Compound UW-ID>]<UW-ID$_2$>
<UW-ID>	:= two alphanumeric characters of `0' – `9' and `A' – `Z'
<Compound UW-ID>	:= two digits of `00' – `99'

For instance, the following shows an example of the list form of a UNL expression of the sentence `I can hear a dog barking outside'.

```
{unl}
[W]
I:01
hear(icl>perceive(agt>person,obj>thing)).@entry:02
dog(icl>canine):03
bark(agt>dog).@entry:04
outside(icl>place):05
:01:06
[/W]
[R]
02aoj01
02obj06
04agt:0103
04plc:0105
[/R]
{/unl}
```

In the above example, between [W] and [/W], UWs 'I', 'hear(icl>perceive(agt>thing,obj>thing))', 'dog(icl>canine)', 'bark(agt>dog),' 'outside(icl>place)' and the scope node `:01' are given a UW-ID from 01 to 06 respectively.

Between [R] and [/R], binary relations are described using the UW-IDs defined in the UW list. For example, `02obj06' in the second line shows that the concept identified by UW-ID 06 is the 'obj' of the concept identified by UW-ID 02. UW-ID 06 means the concept of scope 01, and UW-ID 02 means the concept of 'hear(icl>perceive(agt>thing,obj>thing))'.

Binary relations `04agt:0103' and `04plc:0105' express the UNL expression of scope 01. This is indicated by the CompoundUW-ID `01' described following the relations 'agt' and 'plc'.

qua [":"<Compound UW-ID>] "(" {<UW1>|":" <Compound UW-ID>} "," {<UW2>|":"<Compound UW-ID>} ")"

3.4.1 UWs

3.4.1.1 Syntax of UW

A UW is made up of a character string (an English-language word) followed by a list of constraints. The meaning and function of each of these parts is described in the next section, on Interpretation.

The following is the syntax of description of UWs:

<UW>	::= <headword> [<constraint list>]
<headword>	::= <character>…
<constraint list>	::= "(" <constraint> ["," <constraint>]… ")"
<constraint>	::= <relation label> { `>' | `<' } <UW> [<constraint list>] | <relation label> { `>' | `<' } <UW> [<constraint list>] [{ `>' | `<' } <UW> [<constraint list>]] …
<relation label>	::= "agt" | "and" | "aoj" | "obj" | "icl" | ...
<character>	::= "A" | ... | "Z" | "a" | ... | "z" | 0 | 1 | 2 | ... | 9 | "_" | " " | "#" | "!" | "$" | "%" | "=" | "^" | "~" | "|" | "@" | "+" | "-" | "<" | ">" | "?"

3.4.1.2 Interpretation

HEADWORD

The headword is an English word/compound word/phrase/sentence that is interpreted as a label for a set of concepts: the set made up of all the concepts that may correspond to that in English. A Basic UW (with no restrictions or constraint list) denotes this set. Each Restricted UW denotes a subset of this set that is defined by its constraint list. Extra UWs denote new sets of concepts that do not have English-language labels.

Thus, the headword serves to organize concepts and make it easier to remember which is which.

CONSTRAINTS OR RESTRICTIONS

The **constraint list** restricts the interpretation of a UW to a subset or to a specific concept included within the Basic UW, thus the term `Restricted UWs'. The Basic UW `drink', without a constraint list, includes the concepts of `putting liquids in the mouth', `liquids that are put in the mouth', `liquids with alcohol', `absorb' and others. The Restricted UW 'drink(agt>thing,obj>liquid)' denotes the subset of these concepts that includes `putting liquids in the mouth', which in turn corresponds to verbs such as `drink', `gulp', `chug' and `slurp' in English.

A restriction of a UW is made up of a pair of a relation and a pre-defined UW (or part expression of a pre-defined UW) that holds the relation with this UW. If more than one restrictions are necessary, a comma `,' should be used between restrictions. A Restricted UW is defined through a Master Definition (for details please refer to UW Manual). In a Master Definition, full expressions of pre-defined UWs must be described in the restrictions, whereas as for a UW, if and only if the uniqueness can be kept, part of the pre-defined UWs (its headword or part restrictions) can be used in the restrictions. Relation labels used in the constraint list must be defined in the UNL specifications and should be sorted in alphabetical order if more than one restrictions are used.

In order to define the meaning of a UW more accurately, for instance, a subset concept of UW is always defined under an upper UW that has the closest but more general meaning. This is implemented by linking the UW to be defined with the upper UW using 'icl' relation. For example, UW 'provide(icl>give(agt>thing,gol>thing,obj>thing))' is defined as a subset concept of UW 'give(agt>thing,gol>thing,obj>thing)'. However if the headword of the upper UW is either of `be', `do', `occur' and `uw', such a headword is not necessary to remain in the restrictions of lower UWs as the each set of restrictions of these upper UWs is set enough to restrict their lower UWs. For example, from Master Definition 'drink({icl>do{}agt>thing,obj>liquid{}})' a UW 'drink(agt>thing,obj>liquid)' and a binary relation 'icl(drink(agt>thing,obj>liquid), do(agt>thing,obj>liquid))' are generated. The part related to the headword `do' is removed from its lower UW expression and the binary relation that will be described in the UNLKB shows that 'drink(agt>thing,obj>liquid)' is a subset concept of 'do(agt>thing,obj>liquid)'. For details of description of UW please refer to UW manual.

3.4.1.3 Types of UW

A UW is a character string and most of the UWs are basically made up of an English expression with restrictions. A UW can express various levels' concepts depending on the restrictions and can be used to express a more specific or particular concept or an instance by giving attributes and IDs or restrictions from other UNL expressions. The UWs are divided into four types:

- Basic UWs, which are bare headwords with no constraint list, for example:

```
go
take
house
state
```

- Restricted UWs, which are headwords with a constraint list, for example:

```
state(icl>express(agt>thing,gol>person,obj>thing))
state(icl>country)
state(icl>region)
state(icl>abstract thing)
state(icl>government)
```

- Extra UWs, which are a special type of Restricted UW, for example:

```
ikebana(icl>flower arrangement)
samba(icl>dance)
souffle(icl>food)
```

- Temporary UWs, which are not necessary to define, for example:

```
1234
xyz
```

Basic UWs

Basic UWs are character strings that correspond to English words. Such a basic UW denotes all the concepts that may correspond to those in English. However a basic UW is not used if the English expression is ambiguous. Such a basic UW is usually used as the headwords of Restricted UWs for its various specific concepts. A basic UW is used if the English expression has no ambiguity.

Restricted UWs

Restricted UWs are by far the most important. A Restricted UW is made up of a headword (English expression) with restrictions. It is necessary when the English expression of headword has broader sense (more meanings) than the concept aimed to define. The restrictions restrict the range of the concept that an English expression represents. Each Restricted UW made from an English expression represents a more specific or particular concept, or a subset of the concepts of the English expression.

For example, following are the Restricted UWs made from the English word `state':

- **state((icl>express(agt>thing,gol>person,obj>thing))** is a more specific concept that denotes an action in which humans express something.
- **state(icl>country)** is a more specific sense of `state' that denotes a country.
- **state(icl>region)** is a more specific sense of `state' that denotes a region of a country.
- **state(icl>abstract thing)** is a more specific sense of `state' that denotes a kind of condition that persons or things are in. This UW is defined as a more general concept that can be referred to when defining other synonymous UWs, such as `situation' or `condition'.
- **state(icl>government)** is a more specific sense of `state' that denotes a kind of government.

The information in parentheses is the constraint list and it describes some conceptual restrictions; this is why they are called Restricted UWs. Informally, the restrictions mean `restrict your attention to this particular sense of the word'.

Extra UWs

Extra UWs denote concepts that are not found in English and therefore have to be introduced as extra categories. Foreign-language words are used as headwords using English (Alphabetical) characters.

For example, following are the examples of Extra UWs:

- **ikebana(icl>flower arrangement)** is `a kind of flower arrangement' for the meaning of `something you do with flowers',
- **samba(icl>dance)** is `a kind of dance', and
- **souffle(icl>food)** is `a kind of food'.

To the extent that these concepts exist for English speakers, they are expressed with foreign-language loanwords and do not always appear in English dictionaries. So they simply have to be added to be able to use these specific concepts in the UNL system. The restrictions give the idea of what kind of concept is associated with these Extra UWs and the constraints provide the binary relations between this concept and other, more general, concepts already defined. Needless to say, an Extra UW is also defined through a Master Definition, and a pre-defined UW or its part expression must be used in the restrictions of an Extra UW.

A number or an address of email that has to be used as it is not necessary to define. They can appear in a UNL document and are treated as temporary UWs.

```
ex) 1234
    xyz
```

3.4.2 Compound UWs : Scopes

Compound UWs are a set of binary relations that are grouped together to express a complex concept. A sentence itself is considered as a compound UW. Compound UWs denote complex concepts that are to be interpreted/understood as a whole so that one can talk about their parts all at the same time. A compound UW is expressed by a scope in UNL expressions. A scope makes it possible when a compound UW is necessary to be connected with other UWs.

Consider the following example:

> [Women who wear big hats in movie theaters] should be asked [to leave].

The part of the sentence within square brackets is what should be asked. Only when they are grouped together and considered as a whole unit can the correct interpretation be obtained.

Attributes can be attached to them to express negation, speaker attitudes, etc., which are usually interpreted as modifying the main UW (attached with @entry) and its coordinate UWs within the compound UW (scope).

3.4.2.1 The way to define a Compound UW

A Compound UW is defined by placing a Compound UW-ID immediately after the Relation Label in all of the binary relations that are to be grouped together. Thus, in the example below, `:01' indicates all of the elements that are to be grouped together to define Compound UW number 01.

```
agt:01(wear(aoj>thing,obj>hat), woman(icl>person).@pl)
obj:01(wear(aoj>thing,obj>hat), hat(icl>wear))
aoj:01(big(aoj>thing), hat(icl>wear))
plc:01(wear(aoj>thing,obj>hat), theater(icl>facilities))
mod:01(theater(icl>facilities), movie(icl>art))
agt:01(leave(agt>thing,obj>place).@entry, woman(icl>person).@pl)
```

After this group has been defined, wherever the Compound UW-ID is, for instance `01' in the above example, it can be used to cite the Compound UW. The way to cite a Compound UW is explained in the next section.

A Compound UW is considered as a sentence or sub-sentence, so in the definition of a Compound UW one entry node marked by @entry is necessary.

3.4.2.2 The way to cite a (Compound) UW

Once defined, a Compound UW can be cited or referred to by simply using the Compound UW-ID as an UW. The method is to indicate the Compound UW-ID following a colon `:'. The reference to a Compound UW is also called a Scope Node. The Scope Node has the following syntax:

<Scope Node>	::= ":" <Compound-ID> [<Attribute List>]
<Compound-ID>	::= two digits of a number "01"–"99", except "00"
<Attribute List>	::= { `.' <Attribute> } ...
<Attribute>	::= `@entry' | `@may' | `@past' | ...
<Node>	::= <UW> [`:' <Compound-ID> | <Sentence-ID>]... [<Attribute List>]
<Compound-ID>	::= two digits of a number "01"–"99", except "00"
<Attribute List>	::= { `.' <Attribute> } ...
<Attribute>	::= `@entry' | `@may' | `@past' | ...

To complete the UNL expression of `[Women who wear big hats in movie theaters] should be asked [to leave]', the following are necessary:

```
obj(ask(agt>thing,gol>person,obj>uw).@should.@entry, :01)
gol(ask(agt>thing,gol>person,obj>uw).@should.@entry, woman(icl>person).@pl.@topic)
```

'obj(ask(agt>thing,gol>person,obj>uw).@should.@entry, :01)' shows that Scope 01 is the obj of `ask'.

':01' shows the scope node. It is interpreted as the whole set of binary relations defined above. It means that `:01' should be understood as comprising all of these binary relations. Compound UWs can be cited within other Compound UWs.

3.5 UNL Knowledge Base (UNLKB)

The UNLKB is a semantic network comprising every directed binary relation between UWs. All the binary relations of the UNL KB are in the following format: 'relation(UW1, UW2)=c', where 'c' is the degree of certainty, which has the value 0 (impossible) or 1 to 128 (certain). This binary relation means `UW1 takes UW2 as the relation in certainty value c', or `UW2 plays the role specified by relation for UW1 in certainty

value c'.

The UNLKB Defines Concepts of UWs

The UNLKB provides concepts and UW of the concepts. A UW (Universal Word) is a label for a concept. Concepts labelled by UWs are defined by describing the set of possible relations that each concept can have with other concepts. Definitions of possible relations with other concepts describe behaviours of concepts in terms of other concepts. This behaviour is the property of a concept in the sense that the descriptions of behaviour characterize the concept and provide enough information to understand the semantic structure of the sentence.

The UNLKB Provides Linguistic Knowledge of Concepts

The behaviour of a concept is considered as linguistic knowledge on the concept. This knowledge is used to provide semantic structure of sentences of natural languages. For example, an `author' is a `person', which can take various actions that a person can take such as a person can writes something and something can be a book, and so forth. This level of knowledge is necessary to provide semantic background of natural language sentences. Further knowledge, for example real world knowledge and so forth, will be established based on this linguistic knowledge using the UWs.

UW System

In the UNL KB, all UWs are linked with each other through 'icl',(subclass) 'iof'(element/instance), or 'equ' relations. 'Icl' links a UW of a subclass concept to a super-class concept UW; 'iof' links a UW expressing an instance to a UW of a class concept; The UWs related to each other through 'icl', 'iof' and 'equ' relations make up a hierarchy of UWs. This hierarchy of UWs is the UW system. This UW system allows having multiple super-class concepts. Accordingly, the UW system is a lattice type of network.

The hierarchy of the UW system is constructed by taking the property inheritance and replacement by super-class concept mechanisms into consideration. In UW system, lower UWs inherit the properties of upper UWs; and upper UWs can replace lower UWs to convey a more general sense in the specific context of the lower UWs. All these inheritance and replacement are carried out through the relations 'icl', 'iof' and 'equ'.

In the UNL KB, all possible relations, such as 'agt', 'obj', etc, that an UW can have with others are defined for each. Every possible relation is defined between the two most general UWs of the two categories (of lower UWs) that can have the relation. Utilizing the property inheritance mechanism of the UW system possible relations of lower concepts are deductively inferred and this inference mechanism can reduce the number or binary relations.

Replacement of lower UWs by upper UWs can cause problems by introducing ambiguities if the upper UWs are not close in meaning to the lower UWs. To avoid this, the upper UWs must be the closest UWs among all of the more general UWs . In other word, every UW must be positioned under the closest upper UWs.

The UNLKB defines the syntax and semantics of the language UNL. In the UNL System, UNLKB is used in sentence analysis for disambiguation and in sentence generation for finding more general concepts when encountering unknown concept to the language. The UNLKB also is used to verify UNL expressions since it provide syntax and semantics of UNL.

4. CWL.cdl

The CWL.cdl is a representation of CWL in CDL.nl. The CDL.nl is appropriate for logical treatment of contents together with information in multimedia. The UNL is appropriate for communication beyond language barriers. RDF/OWL is appropriate for dealing with meta data of web pages and has already many applications actually working.

Design Principles

Concept Description Language (CDL) is a language that describes a wide variety of representation media and content, as well as conceptualization of their meaning, in a common format. CDL is a simple extension of the basic principles of XML and is able to inherit stored data tagged with XML.

In contrast to XML, which annotates the document structure (syntactic structure) of content, CDL describes the concept structure (semantic structure) of content. This paper begins with the characteristics of CDL compared to those of XML.

Specifications design for XML is summarized as follows.

1. Tags (document tags) are embedded into content and annotate the document structure. There is basically one type of tag.
2. The document structure model has a nesting tree structure.
3. Tag definitions are attached to annotated content and are mutually referred by name space function.
4. There is style sheet that converts tagged document structure into visual presentation which is easy to understand for humans.

The followings are the key concepts of the CDL specification.

1. Labels (concept labels) are used to describe the concept structure separate from content. Labels are divided largely into entity concept labels and relation concept labels.
2. The concept structure model is a hyper network structure.
3. Label definitions are compiled in CDD (concept definition dictionary) separate from all content targeted for description. The semantic relations between concepts described with labels and the internal structure of the concepts are defined in the CDD. The CDD represents

the ontology of the concepts.
4. There is a grounding mechanism that converts the concept structure into content that is easy to understand for humans.

It is possible to set up partially a mechanism compliant to CDL for XML. This mechanism is realized by RDF/OWL in the Semantic Web. The following are its key characteristics in RDF/OWL specifications design.

1. `Properties' are described.
2. The properties (meta data) of the resources (including content) are described. Description is in the form of a set of triples, each of which consists of subject (resource), property and property value, in frame-like nesting format.
3. Vocabulary for description is organized in ontology. The framework for describing ontology is the RDF Schema, and OWL is the language for describing definitions.

In comparison, the following are the key CDL specifications.

1. `Structures' are described.
2. The concept structure of the entire content is described. The network format consisting of node and arc, each of which is entity concept and the relation concept (and its degenerate form of attribute and attribute value) is created. Furthermore, complex concept type (hyper node and hyper arc) and elemental concept type are created for each concept.
3. Vocabulary for description is organized in the CDDs. The framework for describing vocabulary definitions (ontology) and meta vocabulary to be described are defined a priori in the CDD of CDL.core. The CDD is the concept ontology with Entity, Relation and Attribute as top-level concepts, and the CDD itself is a complex entity.

In contrast to RDF/OWL which was designed in bottom-up design from XML for metadata description, CDL was designed in top-down approach from the basic perspective on concept structure for the purpose of content concept description. For this reason, CDL is consistent, simple and clear in specifications. In CDL implementation, RDF/OWL implementation specifications and XML implementation specifications are prepared.

In CDL, CDL.xxx for various representation media and the various content formats have to be set up, based on CDL.core, which provides the meta-concepts for describing concept definitions. The respective concept vocabularies are defined in their corresponding CDD (CDD.xxx). For natural language media, concept shared with natural language is defined as CDD.nl. Concepts necessary for the conceptualization of meanings representing terms, phrases, sentences and text in natural language are defined. CDLs corresponding to each language, such as Japanese, English, and Chinese, are created as CDL.jpn, CDL.eng, CDL.chi, etc., and CDDs defining concepts for each language are prepared. Furthermore, CDD corresponding to everyday language, CDD for specialized areas and communities, and CDD for controlled language, etc., are created to prepare CDL that corresponds to a variety of language usage.

In addition, CDLs can also be made for mathematical equations, programming languages, video image, music sound, etc., in the form of CDL.math, CDL.prog, CDL.movie, CDL.music, etc. CDDs therefore consist of concepts inherent to each representation media and concepts borrowed from natural language.

In CDL, maintaining continuity with XML is important. This is because the document structure (syntactic structure) serves as a guide for the concept structure (semantic structure) and it is necessary to appropriately preserve the huge amount of content that is XML-tagged. The schemes prepared for appropriate coordination between CDL and XML are following. There are two schemes.

1. **Integration of description and annotation**
   The description function is able to include annotation function. CDL is capable of continuous adaptation from annotation to description, such as all annotation, partly annotation and partly description, and all description.
2. **Implementation specifications on XML**
   In addition to implementation specifications on RDF/OWL, implementation specifications directly on XML will be investigated.

4.1 CDL.core

4.1.1 Model and Syntax Specifications

Model and Syntax determine how concepts or conceptualization are to be modeled and what kind of data format or grammar will be set up to express the model. In order words, it determines the common form of expression for all CDL family languages. Therefore, CDL.core follows this notation format.

First, the basic categories for the model of conceptualization are determined as follows.

```
Concept
    Entity
        ElementalEntity
        ComplexEntity
    Relation
        ElementalRelation
        ComplexRelation
    Attribute
```

Concepts and concept descriptions are modeled in the following network structure, namely:

1. Entities that are to make up the concepts become nodes, and the relations between entities become arcs in the network structure.
2. The nodes and arcs are elemental concepts without structure (ElementalEntity and ElementalRelation) or complex concepts with structure (ComplexEntity and ComplexRelation). Complex concepts correspond to hyper-nodes or hyper-arcs, creating a congestion model for concept structures.
3. Complex concepts classify two types of model for conceptualization in accordance with how relations are connected. In other words, if a relation is connected with the complex concept as a whole, the model determines the whole as a single concept. If a relation is connected

with a concept inside the complex concept, the model determines the concept as a concept modified by the complex concept.
4. Attribute-value pairs are added to the concept description to create model for localized property descriptions of the concept. An attribute is a degenerated Element Relation, and its value is a degenerated Element Entity.

Text Notation

As method for network structure notation for concepts, graph notation with diagram and text notation with symbolic text are made available. Text notation syntax, specifically basic syntax, is shown below. The following four meta symbols are used to describe grammar.

`::='	Left side is defined as right side
`|'	Selection `or'
`*Symbol*'	Meta symbol (Non-terminal symbol)
`…'	Repetition zero or more times

```
ConceptDescription ::= EntityDescription

EntityDescription ::= ElementalEntityDescription | ComplexEntityDescription
ElementalEntityDescription ::=
    {RealizationLabel  DefinitionLabel  AttributeValuePair… ;}
ComplexEntityDescription ::=
    {RealizationLabel  DefinitionLabel  AttributeValuePair… ;
        EntityDescription… RelationDescription… Arc…}

RelationDescription ::=
    ElementalRelationDescription | CompexRelationDescription
ElementalRelationDescription ::=
    {RealizationLabel  DefinitionLabel  AttributeValuePair… ;}
CompexRelationDescription ::=
    {RealizationLabel  DefinitionLabel  AttributeValuePair… ;}
        From To EntityDescription…EntityDescription… Arc…}

AttributeValuePair ::= Attribute =Value
Arc ::= [EntityRealizationLabel-from RelationRealizationLabel EntityRealizationLabel-to]

RealizationLabel ::= #ArbitraryCharacterString
DefinitionLabel ::= DefinitionLabelDefinedInCDD
Attribute ::= AttributeDefinedInCDD
Value ::= ValueDefinedInCDD
```

In both entity and relation, concept description is unified, simple and clear. The difference between elemental concept and complex concept is whether there is a structure description after the delimiter (;). The ComplexRelationDescription includes special ElementalEntity (From and To) to match the inside and the outside of an arc.The reason of the use of "{" and "}" is to perceive element set in structure description as bag (set). Therefore, concept description in structure description will be identified uniquely with realization label. Scope can be set on the realization label according to nested structure of concept description. The scope on the realization lablel is set in open scope and closed scope modes. Open scope is default. Description examples hereafter are shown in open scope. The arcs are in triples with direction from left to right.

There are elemental concept description and complex concept description in concept description. One definition is used in various realizations. Concept description requires localized Attribute-Value pair. These are easy to understand for entity concept but require explanation for relation concept. To understand the explanation, the description in the next and subsequent chapters is necessary. However, it will be summarized below.

1. Complex Relation: Relation concept including RDF/OWL is not intended to have structure. However, any relation does not necessarily remain primitive or uniform, and various levels of relations are used. In such a case, a scheme to describe macro relation with micro relation is extremely useful. For example, description of <employer> and <employee> relation with internal structure describing "someone employs someone" and "someone is employed by someone" is useful.
2. Usefulness of Attribute-Value pair in relation concept: A scheme Attribute-Value pair that describes the modality of relation completeness (necessity or possibility, variety such as whether it is established constantly or what time), relationship with subrelations (that is whether all subrelations are established or some part of subrelations are established), and the certainty factors of relations (at what percentage is establishment possible), etc., will be extremely useful.
3. Meaning of realization of relation concept: For a relation concept with the same name to be used in a variety of locations as concept with different Attribute-Value pairs, a scheme for various realizations of the relation concept is necessary.

In definition of the DefinitionLabel (RealizationLabel in Schema) or RealizationLabel, it is necessary only to specify an arbitrary character string. In reality, notation for complex structure label will be specified in each CDD. For instance, the following complex structure can be expected for Japanese vocabulary concept.

```
DictionaryID-Notation-Reading-PolysemyDiscrimination-Date&TimeOfEntry
```

Graph Notation

Graph notation is a method of expressing a directed graph with entity realization as node and relation realization as arc. Complex Entities are represented as hyper nodes, and Complex Relations in hyper arc. This is illustrated as following figure. Here, ED, ER, RD, and RR mean *EntityDefinitionLabel*, *EntityRealizationLabel*, *RelationDefinitionLabel* , and *RelationRealization Label* respectively. *Attribute-ValuePair* is omitted. It can be described as a graph in which *Attribute* is an arc and *Value* is a node. Also, (*Attribute*=*Value*•••)can be added after the *Realization Label*.

The relation that connects ED* and RD* with ER* and RR* respectively is a realizationOf or a typeOf. When the graph notations of these parts are degenerated, it is simplified as follows.

Visual Notation

Various visual notation methods are available to make text notation clearer. Visual notation methods shown here are typical examples, and other shortcuts can be created for each case.

- **(1) Arc**
  There are shortcuts for arcs and arc groups. The notation

  ```
  [EntityRealizationLabel-from RelationRealizationLabel EntityRealizationLabel-to]
  ```

  is represented as

  ```
  [E1  R  E2]
  ```

  in the following explanation.
- **(1-1)**

  ```
  [E1  R  E2]   [E1−R→E2] or [E2  R−E1] or E1−R→E2 or E2  R−E1
  ```

  However, this notation applies for use of UTF-8. When limited to ASCII code, "-" "->" are used for "–" and "→"
- **(1-2)**
  When *EntityRealizationLabel-from* is common,

  ```
  [E0  R1  E1]      [E0−R1→E1 E0−R2→E2 ••• E0−Rn→En] or
  [E0  R2  E2]       [E0−R1→E1 R2→E2 ••• Rn→En] or
     … E0−R1→E1 R2→E2 … Rn→En
  [E0  Rn  En]
  ```

- **(1-3)**
  When an *EntityRealizationLabel-to* of an arc is equal to an *EntityRealizationLabel-from* of another arc,

  ```
  [E0  R1  E1]    [E0−R1→E1 E1−R2→E2 … En−1−Rn→En] or
  [E1  R2  E2]      [E0−R1→E1−R2→E2 … →En−1−Rn→En] or
    …          E0−R1→E1−R2→E2 … →En−1−Rn→En
  [En-1  Rn  En]
  ```

- **(2) Substitution concept description for realization label inside an arc**
  This is a shortcut for substituting concept description of a realization label for the same realization label inside an arc. If the same realization label is used in two places, substitution is possible only for one realization label. Substitution is possible only between concept description and arc in the same scope.
- **(3) Abbreviation of concept description**
  Abbreviation method of adding realization labels to concept definition labels as suffixes is created. In an element entity description,

  ```
  {RealizationLabel DefinitionsLabel Attribute-ValuePair… ;}
  ```

  can be written as

  ```
  DefinitionLabel RealizationLabel(Attribute-ValuePair…)
  ```

  In a complex entity description,

  ```
  {RealizationLabel DefinitionLabel Attribute-ValuePair… ;EntityDescription… RelationDescription… Arc…}
  ```

  can be written as

  ```
  DefinitionLabel RealizationLabel(Attribute-Value…){EntityDescription… RelationDescription… Arc…}
  ```

  (Example)

  ```
  {#1 Concept a1=v1 a2=v2 a3=v3; ……. }
   Concept#1(a1=v1 a2=v2 a3=v3){ ……. }
  {#1 Concept a1=v1 a2=v2 a3=v3;}
   Concept#1(a1=v1 a2=v2 a3=v3)
  {#1 Concept;}
   Concept#1
  ```

  In an elemental relation description,

  ```
  {RealizationLabel DefinitionsLabel Attribute-ValuePair… ;}
   DefinitionLabel RealizationLabel(Attribute-ValuePair…)
  ```

  In a complex relation description,

  ```
  {RealizationLabel DefinitionLabel Attribute-ValuePair… ; From To
                EntityDescription… RelationDescription… Arc…}
   DefinitionLabel RealizationLabel(Attribute-ValuePair…)
             {From To EntityDescription… RelationDescription… Arc…}
  ```

Examples of CDL Description

Examples of graph notation, text notation and visual notation are shown here.

[EXAMPLE 1]

- **Graph notation:**

- **Text notation:**

```
{#a A;
{#x1 A1;}{#x2 A2;}{#x3 A3;}
{#y1 R1;}{#y2 R2;}{#y3 R3;}
[#x1 #y1 #x2]
[#x2 #y2 #x3]
[#x3 #y3 #x1]}
```

- **Visual notation:**
  When (1-3) is applied,

```
{#a A;
{#x1 A1;}{#x2 A2;}{#x3 A3;}
{#y1 R1;}{#y2 R2;}{#y3 R3;}
[#x1-#y1→#x2-#y2→#x3-#y3→#x1]}
```

  When (2) is applied,

```
{#a A;
  {#x1 A1;}-{#y1 R1;}→{#x2 A2;}-{#y2 R2;}→{#x3 A3;}-{#y3 R3;}→#x1}
```

  When (3) is applied,

```
A#a{A1#x1-R1#y1→A2#x2-R2#y2→A3#x3-R3#y3→#x1}
```

[EXAMPLE 2]

- **Graph notation:**

- **Text notation:**

```
{#b B;
  {#b1 B1;}
  {#a A;
    {#x1 A1;}{#x2 A2;}{#x3 A3;}
    {#y1 R1;}{#y2 R2;}{#y3 R3;}
    [#x1 #y1 #x2]
    [#x2 #y2 #x3]
    [#x3 #y3 #x1]}
  {#b2 B2;}
  {#z1 S1;}{#z2 S2;}{#z3 S3;}
  [#b1 #z1 #a]
  [#a #z2 #b2]
  [#b2 #z3 #b1]
  [#b2 #z3 #x3]}
```

- **Visual notation:**
  When (1), (2) and (3) are applied,

```
B#b {B1#b1-S1#z1→A#a{A1#x1-R1#y1→A2#x2-R2#y2→A3#x3-R3#y3→#x1}-S2#z2→B2#b2-S3#z3→#b1, #b2-#z3→#x3}
```

Extended Model and Syntax

The form of concept description in 2.1 is the basic form. That is to use conceptual definitions in CDD for realizations in concept description. The following explains the special realization forms and the concept description forms without concept definitions which are classified as Extended Model and Syntax.

(1) CONCEPT IN WHICH CONTENT ITSELF INCLUDES MEANING (TEXTUAL CONCEPT)

This indicates a concept in which the content itself includes meaning, where the definition label of the concept is expressed by the content (content skeleton) itself. It defines the expression of a concept that is recalled when a standard audience reads, listens to or sees the content. Here, a piece of text in natural language will be used as a typical example of content.

A concept recalled when a standard audience whose mother tongue is the language used in the text description reads *text* is expressed as *text* surrounded by "<" and ">".

```
{RealizationLabel <Text> Attribute-ValuePair… ;}
```

The corresponding visual notation is shown below. However, *Attribute-ValuePair* is omitted.

```
<RealizationLabel; Text>
```

or

```
<Text>RealizationLabel
```

If *RealizationLabel* is Nil(`#'only)

<*Text*>

(Example)

```
{#1 <concept description>;}
   <#1;concept description>  or  <concept description>#1
{# <concept description>;}
   <concept description>
```

[Note]

Here, the role or function of the textual concept was explained by placing text in the definition label of the concept description. Another alternative in explanation is to place the text in the structure description of a complex concept. With this method, a nested structure of textual concepts can be created. Specifically, a textual concept is shown as a complex entity as follows.

```
{RealizationLabel TextConcept Attribute-ValuePair… ; <Text>}
```

(2) Content surface notation as concept (Literal Concept)

A surface notation of the content itself can be regarded a concept. Here, text in natural language (text piece) will be used as a typical example of content.

Text with surrounded by ' and ' expresses the literal concept.

```
{RealizationLabel 'Text ' Attribute-ValuePair… ;}
```

Corresponding visual notation is shown below. However, *Attribute-ValuePair* is omitted.

```
'RealizationLabel;Text '
```

or

```
'Text 'RealizationLabel
```

If *RealizationLabel* is Nil(`#'only)

```
'Text'
```

(Example)

```
{#a 'Concept description language';}
    '#a;Concept description language' or 'Concept description language'#a
{# 'Concept description language';}
    'Concept description language'
```

(3) Temporary concept (or anonymous concept)

This concept is used temporarily only for description of realization. *DefinitionLabel* is made identical to *RealizationLabel* in order to express a concept on a temporary basis. It is very often used to express complex concepts in temporary as follows,

```
{RealizationLabel RealizationLabel Attribute-ValuePair… ;
                    EntityDescription… RelationDescription… Arc…}
```

The corresponding visual notation is shown below. However, *Attribute-ValuePair* is omitted.

```
{RealizationLabel ; EntityDescription… RelationDescription… Arc…}
```

or

```
{EntityDescription… RelationDescription… Arc…}RealizationLabel
```

If *RealizationLabel* is Nil(`#'only)

```
{EntityDescription… RelationDescription… Arc…}
```

(Example)

```
{#a #a; ……. }
   {#a; ……. } or { ……. }#a
{# #; ……. }
   { ……. }
```

(4) Concept with only one realization (Proper Concept)

This expresses concept that has only one realization of concept definition. Most proper nouns and relation concepts correspond to this. Definition label itself becomes the realization label to express that it is a unique concept.

```
{RealizationLabel DefinitionLabel Attribute-ValuePair… ;}
```

The corresponding visual notation is shown below. However, *Attribute-ValuePair* is omitted.

```
    DefinitionLabel
```

(Example)

```
{Tokyo_Station Tokyo_Station;}
   Tokyo_Station
{Mt_Fuji Mt_Fuji;}
   Mt_Fuji
{John John;}
   John
```

(5) Concept in which the realization notation itself is the primary concept definition (Identifier Concept)

This concept has a realization notation and a definition that match each other. This concept applies to numbers and other identifiers. The definition label has # removed from the *RealizationLabel* in order to express the identifier concept.

```
{RealizationLabel RealizationLabelWithout# Attribute-ValuePair… ;}
```

The corresponding visual notation is shown below. However, *Attribute-ValuePair* is omitted.

```
    RealizationLabelWithout#
```

(Example)

```
{#256 256;}
   256
{#ISBN9-876-54321-0 ISBN9-876-54321-0;}
   ISBN9-876-54321-0
```

Concept Definition and Realization

Concept description is divided largely into concept definition description (Definition or Concept Definition) and concept realization description (Realization or Concept Realization). Definition description is called concept schema because of the description framework, and realization may be called instantiation.

In the object-oriented approach in programming languages, etc., the terms Class Definition and Instantiation are commonly used. In CDL, the word Realization is used instead of Instantiation. Instantiation yields an example Instance that satisfies a Definition. In the object-oriented approach, Instance is used as a single, independent case. In Concept Description under CDL, Realizations are not used independently and used in a variety of cases, with various roles given in Concept Description. They may be, for instance, as generic, unspecified, or specified conceptual examples. In order to express such uses appropriately, the term Realization is used. However, in terms of the basic scheme of implementation, Realization stemming from Definition roughly corresponds to Instanciation.

Concept definition descriptions are compiled in CDD (Concept Definition Dictionary). CDD directly reflects the respective conceptualization model. And, all CDDs are detailed from common conceptualization model to their conceptualization models. In other words, all CDD.xxx corresponding to CDL.xxx share the following top-level concept hierarchies.

```
Concept
    Entity
        ElementalEntity
        ComplexEntity
    Relation
        ElementalRelation
        ComplexRelation
    Attribute
```

The relation between Concept Definition and Concept Realization is illustrated as follows.

(Example)

An example of Definition and Realization follows. Please note that this shows the structure only, and some details have been omitted. The following is an example of Realization of the Concept Description corresponding to the English-language example in Appendix 1, Example 1 shown as Text Concept. For the sake of simplification, the definition labels of the concept (#buy, #report, #computer, etc.) are shown in the heading only. In practice, a scheme for identifying multiple definitions is necessary for definition labels in case there are multiple meanings.

<John reported to Alice that he bought a computer yesterday>

- **Definition:**

```
Concept
    Entity
        ElementalEntity
            NominalEntity
                {#computer Schema type='ElementalEntity'  …; …}
                {#John Schema type='ElementalEntity'  …; …}
                {#Alice Schema type='ElementalEntity'  …; …}
                {#yesterday Schema type='ElementalEntity'  …; …}
            VerbalEntity
                {#buy Schema type='ElementalEntity'  …; …}
```

```
                {#report Schema type='ElementalEntity'  …; …}
        ComplexEntity
            {#Event Schema type='complexEntity'  …; …}
    Relation
        ElementalRelation
            {#agt Schema type='ElementalRelation'  …; …}
            {#obj Schema type='ElementalRelation'  …; …}
            {#cnt Schema type='ElementalRelation'  …; …}
            {#seq Schema type='ElementalRelation'  …; …}
        ComplexRelation
    Attribute
        {#AttributeOfEvent Schema type='Attribute'  …; …}
        {#AttributeOfNominalEntity Schema type='Attribute'  …; …}
```

- **Realization:**

```
{#A Event temporality='past';
  {#agt agt} {#obj obj} {#tim tim} {#cnt cnt} {#seq seq}
    {#B Event temporality='past';
      {#b1 buy;}
      {#b2 computer realizationMode='def';}
      {#b3 yesterday realizationMode='def';}
      [#b1 #agt John] [#b1 #obj #b2] [#b1 #tim #b3]}
    {John John;}
    {#a1 report realizationMode='def';}
    {Alice Alice;}
    [#a1 #agt John] [#a1 #obj Alice] [#a1 #cnt #B] [#A #seq #B]}
```

There are cases of multiple CDDs used for a single Realization Description. An example is illustrated below.

(Example)

Three CDDs, namely CDD.nl, CDD.eng and CDD.jpn, are used for the Realization Description in this example. In the Realization Rescription, CDD.nl is used as the principal dictionary. Concept labels other those taken from the principal dictionary have dictionary identifiers attached as prefixes. The prefix is "*DictionaryIdentifier*:"

The following is an example of Realization of the Concept Description in Appendix 1, Example 1 shown as Text Concept employing the English and Japanese examples.

<<John reported to Alice that he bought a computer yesterday>>

- **Definition:**
  [CDD.nl]

```
Concept.nl
    Entity
        ElementalEntity
        ComplexEntity
            {#Event Schema type='ComplexEntity'  …; …}
    Relation
        ElementalRelation
            {#agt Schema type='ElementalRelation'  …; …}
            {#obj Schema type='ElementalRelation'  …; …}
            {#tim Schema type='ElementalRelation'  …; …}
            {#cnt Schema type='ElementalRelation'  …; …}
            {#seq Schema type='ElementalRelation'  …; …}
        ComplexRelation
    Attribute
        {#AttributeOfEvent Schema type='Attribute'  …; …}
        {#AttributeOfNominalEntity Schema type='Attribute'  …; …}
```

  [CDD.eng]

```
Concept.eng
    Entity
        ElementalEntity
            NominalEntity
                {#computer Schema type='ElementalEntity'  …; …}
                {#John Schema type='ElementalEntity'  …; …}
                {#Alice Schema type='ElementalEntity'  …; …}
                {#yesterday Schema type='ElementalEntity'  …; …}
            VerbalEntity
                {#buy Schema type='ElementalEntity'  …; …}
                {#report Schema type='ElementalEntity'  …; …}
        ComplexEntity
    Relation
        ElementalRelation
        ComplexRelation
    Attribute
```

- **Realization:**

```
{#; {#agt agt} {#obj obj} {#tim tim} {#cnt cnt} {#seq seq}
  {#A1 Event temporality='past';
    {#B1 Event temporality='past';
      {#b11 CDD.eng:buy;}
      {#b12 CDD.eng:computer realizationMode='def';}
      {#b13 CDD.eng:yesterday realizationMode ='def';}
      [#b11 #agt CDD.eng:John] [#b11 #obj #b12] [#b11 #tim #b13]}
    {CDD.eng:John CDD.eng:John;}
    {#a11 CDD.eng:report realizationMode='def';}
    {CDD.eng:Alice CDD.eng:Alice;}
    [#a11 #agt CDD.eng:John] [#a11 #obj CDD.eng:Alice] [#a11 #cnt #B1] [#A1 #seq #B1]}
```

Schema Specifications

This chapter establishes schema and other meta-concepts to make up language specifications for describing concept definitions. In other words, this comprises the meta-language specifications to describe the content of CDDs for all CDL family languages (CDLs). Meta-vocabularies and meta-sentence structures in the meta-language specifications are defined in the CDD.core which is a concept definition dictionary for CDL.core. CDD.core definition is assumed to be a priori and have no higher meta level definition. The relation between CDD.core and CDD.xxx is that CDD.xxx is a Realization Description from the Definition Description by CDD.core.

The relation between CDD.core and CDD.xxx is shown in more detail as follows.

Basically, all concepts in CDD.xxx (Entity, Relation and Attribute) are realized as Entity in CDL.core. In other words, everything becomes entity concept (objects). CDL.core words, that is Entities, Relations and Attributes in CDL.core, are used to express Concept Definition Description for all CDL.xxx concepts as entities. The sentence pattern for the description is the Schema. It is the ComplexEntity of CDLs.core.

The meta description mechanism for CDD Schema is organized as follows.

1. Definition description in CDD.core

   ```
   {#Schema Schema … ; … }
   ```

2. Definition description in CDD.xxx: Realization description with Schema of CDL.core

   ```
   {#DefinitionLabel Schema … ; … }
   ```

3. Realization description with CDL.xxx

   ```
   {#RealizationLabel DefinitionLabel … ; … }
   ```

CDD.core

The CDD of CDLs.core (CDD.core) is as follows. These meta concepts are explained in subsequent sections.

```
Concept.core
    Entity
        ElementalEntity
            QuoteExpression
            QuoteCategory
            FromComplexRelation
            ToComplexRelation
        ComplexEntity
            ConceptDefinitionDictionary
            Schema
            Constraint
            Arcs
            Structure
            Example
            Explanation
    Relation
        ElementalRelation
            union
            intersection
            complement
            attribute
            value
            default
            relationFrom
            relationTo
            subConceptOf
                subEntityOf
                subRelationOf
                subAttributeOf
        ComplexRelation
    Attribute
```

The Definition Description for the concept definition dictionary CDD.xxx is *ConceptDefinitionDictionary*, and the complex entities for CDD.xxx are the *Schema*. Complex concepts such as *Structure* and *Constraint* are elements of *Schema*. Furthermore, *QuoteExpression* and *QuoteCategory* are elements of *Structure* and *Constraint*.

Quotation

For Definition Description of Schema, reference of other Schema that have been already defined is necessary. This reference is not reference by Schema Realization but by Quotation as is. Two forms are created in quotation. One is Quotation of object as Expression and the other is Quotation of object as Category. Also, the quotation itself is an Entity.

Expression quote

In this form, an object is quoted as an expression. The quotation can be replaced with the quoted schema. The current approach to quotation is to quote the schema in question as is. Eventually, it will be necessary to replace some parts of the schema in a quotation. In other words, part

of the quoted schema must be replaced in the quotation, using expressions that are suited for each quotation. For this purpose, it is necessary to have a function designating the location of schema description for replacement and a function for instructing what kind of replacement should be at the location.

```
QuoteExpression ::= {RealizationLabel QuoteExpression
                                        type=TypeOfQuotedSchema
                                        schema=DefinitionLabelOfQuotedSchema
                                        mode=QuotationMode
                                        usage=UsageManner
                                        minCardinality= MinimumCardinality
                                        maxCardinality= MaximumCardinality ;}

TypeOfQuotedSchema::= 'entity' | 'relation' | 'attribute' (Default: 'entity')
DefinitionLabelOfQuotedSchema::=DefinitionLabel
QuotationMode::= 'aConcept' | 'theConcept' (Default: 'aConcept')
UsageManner::='surely' | 'possibly'  (Default: 'surely')
MinimumCardinality::=Number  (Default:1)
MaximumCardinality::=Number | 'any'  (Default:1)
Number::= 0 | 1 | 2 | 3 | 4 | 5 | ……
                          However, MinimumCardinality ≤ MaximumCardinality
```

type specifies the type of quoted schema realization. *schema* specifies the concept label of quoted schema realization. *mode* specifies the scope of concept from which the quote derives. In other words,

1. It specifies the concept indicated by the quote or the concept to which the concept indicated by the quote can be traced ('aConcept').
2. (2) It specifies that the quote is a quote of the concept indicated ('theConcept').

usage specifies whether the quote is used without fail ('surely') or might not be used ('possibly'). Frequency of quote is specified by minimum cardinality (minCardinality) and maximum cardinality (maxCardinality). If the MinimumCardinality and Maximum Cardinality have the same quoted number, the quotation occurrs as the quoted number. Maximum cardinality can be made arbitrary range ('any').

[Note]

There is another approach in defining quote expressions. *DefinitionLabelOfQuotedSchema* can be placed in a QuoteExpression. In order to indicate it is quote expression, the double quotation is prefixed to the *QuotedSchemaDefinitionLabel*. The definition form is as follows.

```
QuoteExpression ::= {RealizationLabel  "DefinitionLabelOfQuotedSchema
                                       type=TypeOfQuotedSchema
                                       mode=QuotationMode
                                       usage=UsageManner
                                       minCardinality=MinimumCardinality
                                       maxCardinality=MaximumCardinality ;}
```

(Example)

Quotation of #Event schema

```
{#e1 QuoteExpression type='entity' schema=#Event usage='possibly' minCardinality="0"
    maxCardinality='any' ;}
```

When visual notation shortcut is used,

```
 "#Event#e1(usage='possibly' minCardinality="0" maxCardinality='any')
```

or

```
 "Event#e1(usage='possibly' minCardinality="0" maxCardinality='any')
```

[Visual notation]

Using the shortcut in concept description indicated in 2.3 (3), a quote expression will be shown as follows in visual notation.

```
{RealizationLabel QuoteExpression
                  type=TypeOfQuotedSchema
                  schema=DefinitionLabelOfQuotedSchema
                  mode=QuotationMode
                  usage=UsageManner
                  minCardinality=MinimumCardinality
                  maxCardinality=MaximumCardinality ;}
 QuoteExpression RealizationLabel (type=QuotedSchemaType
                                   schema=DefinitionLabelOfQuotedSchema
                                   mode=QuotationMode
                                   usage=UsageManner
                                   minCardinality=MinimumCardinality
                                   maxCardinality=MaximumCardinality)
```

Furthermore, the following shortcut for abbreviation is defined.

```
 "DefinitionLabelOfQuotedSchema RealizationLabel (type=QuotedSchemaType
                                                  mode=QuotationMode
                                                  usage=UsageManner
                                                  minCardinality=MinimumCardinality
                                                  maxCardinality=MaximumCardinality)
```

Category quotation

In this form, the object is quoted as a category. A category expressed by the quoted schema, that is, a set, is quoted. Set operations union, intersecton, and complement can be applied on the quoted category.

```
QuoteCategory ::=
```

```
    {RealizationLabel QuoteCategory schema= DefinitionLabelOfQuotedSchema;} |
    {RealizationLabel QuoteCategory; QuoteCategory-a QuoteCategory-b
        [RealizationLabel-a union RealizationLabel-b]} |
    {RealizationLabel QuoteCategory; QuoteCategory-a QuoteCategory-b
        [RealizationLabel-a intersection RealizationLabel-b]} |
    {RealizationLabel QuoteCategory; QuoteCategory-a QuoteCategory-b
        [RealizationLabel-a complement RealizationLabel-b]}

DefinitionLabelOfQuotedSchema::=DefinitionLabel
```

schema specifies the concept label of the quoted schema realization. Three relation concepts are provided for operators between category sets. They are operator for union of category sets (union), operator for identifying common sets (intersection), and operator for complementing sets (complement).

[Note]

There is another approach for defining category expressions. QuoteCategory can be replaced by *DefinitionLabelOfQuotedSchema*. In order to indicate it is a quote category, the ? is prefixed to the *DefinitionLabelOfQuotedSchema*. The definition form is as follows.

```
QuoteCategory ::=
    {RealizationLabel  ? DefinitionLabelOfQuotedSchema;} |
    {RealizationLabel ; QuoteCategory-a QuoteCategory-b
        [RealizationLabel-a union RealizationLabel-b]} |
    {RealizationLabel ; QuoteCategory-a QuoteCategory-b
        [RealizationLabel-a intersection RealizationLabel-b]} |
    {RealizationLabel ; QuoteCategory-a QuoteCategory-b
        [RealizationLabel-a complement RealizationLabel-b]}
```

[Visual notation]

Using the shortcut in concept description indicated in 2.3 (3), quote category will be shown as follows in visual notation. However, this case does not include set operations.

```
{RealizationLabel QuoteCategory
    schema=DefinitionLabelOfQuotedSchema;}
 QuoteCategoryRealizationLabel (schema= DefinitionLabelOfQuotedSchema)
```

Furthermore, the following shortcut for abbreviation is defined.

```
 ? DefinitionLabelOfQuotedSchema RealizationLabel
```

If category does not rely on Realization,

```
 ? DefinitionLabelOfQuotedSchema
```

(Example)

In the animals that fly, a category defined as the union of birds (excluding penguins and ostritches) and bats is shown in the following category quote.

```
{#3 QuoteCategory; {#2 QuoteCategory; {#1QuoteCategory; [?penguin union ?ostrich]}
                       [?bird complement #1]}
                    [#2 union ?bat]}
```

Furthermore, this can be abbribiated as:

```
 ?{[?{[?bird complement ?{[?penguin union ?ostrich]}]}union ?bat]}
```

or

```
 {[{[?bird complement {[?penguin union ?ostrich]}]}union ?bat]}
```

Constraint

If a concept for definition is a component of a complex concept, Constraint is a complex entity used for defining structural restrictions of the component in the complex concept. Constraint is a necessary condition, not a sufficient condition.

```
Constraint ::= {RealizationLabel Constraint at=WhereConstraintIsToBeSatisfied;
                                 QuoteCategory…
                                 EntityArc… }

EntityArc ::= {RealizationLabel Arcs ; Arc… }

WhereConstraintIsToBeSatisfied;::= ComplexConceptLabel
Arc ::= [Label-1 relationFrom Label-2][Label-2 relationTo Label-3]
Label-1 ::= RealizationLabelForQuoteCategoryOfEntity
Label-2 ::= RealizationLabelForQuoteCategoryOfRelation
Label-3 ::= RealizationLabelForQuoteCategoryOfEntity
```

`at=' specifies the location of a complex concept where the target object(component of the complex concept) satisfies this constraint. The complex concept can be an Entity in some cases or Relation in other cases. *Constraint* is described with multiple *QuoteCategory*s and *EntityArc*s. *EntityArc* specifies *Arc* as entity. *Arcs* in *ComplexEntity* describes Conditions that must be satisfied by pairs of *QuoteCategory*s. If multiple *EntityArc*s are present, at least one of the constraints specified by the *EntityArc*s should be satisfied. If multiple *Arc*s are described in the *EntityArc*, all of the constraints specified by these arcs must be satisfied.

Constraint on an *Entity* is described by enumerating to what *Entity*s this *Entit*y must be connected and to what *Relation*s. *Constraint* on a *Relation* is described by enumerating to what *Entity*s this *Relation* must be connected.

(Example)

In the following, *Constraint* on a complex event #Event are described. `Specifically, what can be "agt (agent)" of "fly" is the set in which "penguin" and "ostrich" have been removed from "bird", and "bat" has been added. And, the possible "plc (place)" for "fly" is "sky.'"

Description of Constraint is shown below in the form as it is.

```
{#c Constraint at=#Event;
  {#1 QuoteCategory schema=#penguin;}
  {#2 QuoteCategory schema=#ostrich;}
  {#3 QuoteCategory schema=#bird;}
  {#4 QuoteCategory schema=#bat;}
  {#5 QuoteCategory schema=#sky;}
  {#6 QuoteCategory schema=#fly;}
  {#7 QuoteCategory schema=#agt;}
  {#8 QuoteCategory schema=#plc;}
  {#i QuoteCategory; [#1 union #2]}
  {#j QuoteCategory; [#3 complement #i]}
  {#k QuoteCategory; [#j union #4]}
  {#a Arcs; [#6-relationFrom->#7][#7-relationTo->#k]
            [#6-relationFrom->#8][#8-relationTo->#5]}}
```

Visual notation is used for simplification as follows.

```
{#c Constraint at=#Event;
  {#a Arcs; [?fly-relationFrom->?agt-relationTo->
                ?{[?{[?bird complement ?{[?penguin union ?ostrich]}]}union ?bat]}]
              [?fly-relationFrom->?plc-relationTo->?sky]}}
```

(Example)

Constraint that satisfies the Relation "agt (agent)" is described as follows. `Namely, whether "agt(agent)" represents Relation between VerbalEntity "Do" and "NominalEntity" or Relation between NominalEntity "Action" and "NominalEntity.'"

Description of Constraint is shown below in the form as it is.

```
{#c Constraint at=#Event;
  {#1 QuoteCategory schema=#agt;}
  {#2 QuoteCategory schema=#Do;}
  {#3 QuoteCategory schema=#NominalEntity;}
  {#4 QuoteCategory schema=#Action;}
  {#5 QuoteCategory schema=#NominalEntity;}
  {#a Arcs; [#2-relationFrom->#1][#1-relationTo->#3]}
  {#b Arcs; [#4-relationFrom->#1][#1-relationTo->#5]}}
```

Visual notation is used for simplification as follows.

```
{#c Constraint at=#Event;
  {#a Arcs; [?Do-relationFrom->?agt-relationTo->?NominalEntity]}
  {#b Arcs; [?Action-relationFrom->?agt-relationTo->?NominalEntity]}}
```

Structure

Structure is a complex entity used to define the structure of complex concepts. This structure description is a necessary condition, not a sufficient condition.

```
Structure ::= {RealizationLabel Structure type=TypeOfComplexConcept;
                                QuoteExpressionOrLocalConceptDefinition…
                                EntityArc… }

TypeOfComplexConcept; ::= 'entity' | 'relation' | 'attribute'  (Default: 'entity')
QuoteExpressionOrLocalConceptDefinition::= QuoteExpression | LocalConceptDefinition
```

`type=' specifies whether the structure definition is for a complex entity ('entity'), a complex relation ('relation') or an attribute ('attribute'). Entities and relations making up the complex cocept are enumerated. Enumeration may be in expression quote or local definition description. Local definition description is as shown in section 2.5 and is a concept definition description that does not employ CDD. Condition of the Structure that must satisfy entities and relations enumerated is described in the multiple EntityArcs.

The aforementioned Structure definition and Constraint definition are available in basic definition function only in CDL.coer Specifications Version 1. Definition function will be extended with attention to implementation method.

(1) Structure definition of complex entity

```
{RealizationLabel Structure type='entity';
                  QuoteExpressionOrLocalConceptDefinition…
                  EntityArc… }
```

(Example)

Structure definition of single event (corresponds to simple sentence)

```
{#s Structure type='entity';
  {#1 QuoteExpression
      schema=#VerbalEntity;}
  {#2 QuoteExpression
      schema=#NominalEntity minCardinality="1" maxCardinality='any' ;}
  {#3 QuoteExpression
      type='relation' schema=#CaseRelation minCardinality="1" maxCardinality='any';}
  {#a Arcs; [#1-relationFrom->#3][#3-relationTo->#2]}}
```

Simplifying with visual notation

```
{#s Structure type='entity';
  "VerbalEntity#1
  "NominalEntity#2(minCardinality="1" maxCardinality='any')
  "CaseRelation#3(type='relation' minCardinality="1" maxCardinality='any')
  {#a Arcs; [#1-relationFrom->#3-relationTo->#2]}}
```

(2) Structure definition for complex relation

```
{RealizationLabel Structure type='relation';
                  QuoteExpressionOrLocalConceptDefinition…
                  FromConceptQuote
                  ToConceptQuote
                  EntityArc… }

FromConceptQuote::=
    {RealizationLabel QuoteExpression schema=#FromComplexRelation;}
ToConceptQuote::=
    {RealizationLabel QuoteExpression schema=#ToComplexRelation;}
```

(Example)

This example defines the "Employer" relation as a complex relation.

```
{#s Structure type='relation';
  {#1 QuoteExpression schema=#employ;}
  {#2 QuoteExpression schema=#ToComplexRelation;}
  {#3 QuoteExpression schema=#FromComplexRelation;}
  {#4 QuoteExpression type='relation' schema=#agt;}
  {#5 QuoteExpression type='relation' schema=#obj;}
  {#a Arcs; [#1-relationFrom->#4][#4-relationTo->#2]
  [#1-relationFrom->#5][#5-relationTo->#3]}}
```

Simplifying with visual notation

```
{#s Structure type='relation';
  "employ#1
  "ToComplexRelation#2
  "FromComplexRelation#3
  "agt#4(type='relation')
  "obj#5(type='relation')
  {#a Arcs; [#1-relationFrom->#4][#4-relationTo->#2]
            [#1-relationFrom->#5][#5-relationTo->#3]}}
```

(3) Structural definition of attribute

```
{Realization label Structure type='attribute';
                   QuoteExpressionOrLocalConceptDefinition…
                   EntityArc… }
EntityArc ::= {RealizationLabel Arcs ; Arc… }
Arc ::= [Label-1 attribute Label-2][Label-2 value Label-3]
          [Label-2 default Label-3]
Label-1 ::= RealizationLabelForQuoteExpressionOfEntity
Label-2 ::= RealizationLabelForQuoteExpressionOfRelation
Label-3 ::= RealizationLabelForQuoteExpressionOfEntity
```

This structural definition assumes that set of attribute-value pairs is a degenerate complex concept. In *QuoteExpressionOrLocalConceptDefinition* which specifies structure elements, only local definition description is used. Three Relations (attribute, value, default) are used for entity arcs.

(Example)

This example defines Structure description for Event attributes (temporality, aspectuality and aspectModify).

```
{#s Structure type='attribute'
  {temporality temporality;}
  {#temporalityValue; 'past' 'present' 'future' 'permanent'}
  {aspectuality aspectuality;}
  {#aspectualityValue; 'begin' 'complete' 'continue' 'custom' 'end'
    'experience' 'progress' 'repeat' 'state' 'unspec'}
  {aspectModify aspectModify;}
  {#aspectModifyValue; 'just' 'soon' 'yet' 'unspec'}
  {#a Arcs;
    [#Event-attribute->temporality]
    [temporality-value->#temporalityValue]
    [temporality-default->'present']
    [#Event-attribute->aspectuality]
    [aspectuality-value->#  aspectualityValue]
    [aspectuality-default->'unspec']
    [#Event-attribute->aspectModify]
    [aspectMody-value->#  aspectModifyValue]
    [aspectModify-default->'unspec']}}
```

When simplified in visual notation,

```
 {#s Structure type='attribute'
  {#a Arcs;
    [#Event-attribute->temporality]
    [temporality-value->{'past' 'present' 'future' 'permanent'}]
    [temporality-default->'present']
    [#Event-attribute->aspectuality]
    [aspectuality-value->{'begin' 'complete' 'continue' 'custom'
```

```
      'end' 'experience' 'progress' 'repeat' 'state' 'unspec'}]
    [aspectuality-default->'unspec']
    [#Event-attribute->aspectModify]
    [aspectMody-value->{'just' 'soon' 'yet' 'unspec'}]
    [aspectModify-default->'unspec']}}
```

Schema

A Schema is a complex entity used in concept definition description. The definition is described by the Structure and the Constraint which were explained above sections.

```
Schema ::= {#DefinitionLabel Schema type=TypeOfConcept;
                                         AttributeDefinition
                                         Structure
                                         Constraint
                                         Explanation
                                         Examples
                                         Arc…}
TypeOfConcept ::= 'ElementalEntity' | 'complexEntity' | 'ElementalRelation' |
                  'complexRelation' | 'attribute'
                  (Default: 'ElementalEntity')
AttributeDefinition ::= QuoteExpressionOfAttributeTypeSchema |
                        StructureOfAttributeDefinition
```

The Schema type (type) specifies the type of concept in a definition description. The structure of definition consists of Attribute Definition, Structure, Constraint, Explanation, Examples, and an arbitrary number of Arcs. The Attribute Definition describes the attribute-value set belonged to the concept to be defined. The Structure describes the structure of the complex concept, if the type of the concept to be defined is a complex one. The Constraint describes the constraint in the structure of a complex concept if the concept to be defined is a component of the complex concept. The Explanation and the Example are secondary information for the Schema and omitted here. The Arcs describe the semantic relation with other concepts as arc information. Also, the Element Relations that constitute Relation are embedded in the Arcs. The following explanation will be limited to the description of a relation corresponding to subConceptOf.

(1) Elemental Entity Type Schema

```
{#DefinitionLabel Schema type='ElementalEntity';
                         AttributeDefinition
                         Constraint
                         Arc…}

Arc ::= [#DefinitionLabel subEntityOf #DefinitionLabelOfSuperiorConcept]
```

(Example)

This is a definition for NominalEntity. The Attribute Definition uses an Attribute type Schema (the Example 1 in the following (5)) as a quoted expression, and the Constraints must be satisfied such as the NominalEntity is connected to ?VerbalEntity and ?caseRelation in the complex entity (Event) and must be described as subConcept of #ElementalEntity.

```
(#NominalEntity Schema type='elemntalEntity';
  "AttributeOfNominalEntity#s
  {#c1 Constraint at=#Event;
  {#a1 Arcs; [?VerbalEntity-relationFrom->?caseRelation
     -relationTo->#NominalEntity]}}
  [#NominalEntity subEntityOf #ElementalEntity]}
```

(2) Complex entity type schema

```
{#DefinitionLabel Schema type= 'complexEntity';
                         AttributeDefinition
                         StructureDefinitionOfComplexEntity
                         Constraint
                         Arc…}

Arc ::= [#DefinitionLabel subEntityOf #DefinitionLabelOfSuperiorConcept]
```

(Example)

A definition of a single event (Event) is used as an example. Attribute is described by quoting the attribute-type schema definition (the Example 2 in the following (5)).

```
{#Event Schema type= 'complexEntity';
  "AttributeOfEvent#q
  {#s1 Structure type='entity';
    "VerbalEntity#1
    "NominalEntity#2(minCardinality="1" maxCardinality='any')
    "caseRelation#3(type='relation' minCardinality="1" maxCardinality='any')
    {#a1 Arcs; ["VerbalEntity#1-relationFrom->"caseRelation#3
      -relationTo->?NominalEntity#2]}}
  {#c1 Consraint at=#Situation;
    {#a2 Arcs; [#Event-relationFrom->?interEventRelation
      -relationTo->?Event]}
    {#a3 Arcs; [?Event -relationFrom->?interEventRelation
      -relationTo->#Event]}}
  [#Event subEntityOf #ComplexEntity]}
```

(3)Elemental Relation Type Schema

```
{#DefinitionLabel Schema type='ElementalRelation';
                        AttributeDefinition
                        Constraint
                        Arc…}

Arc ::= [#DefinitionLabel subRelationOf #DefinitionLabelOfSuperiorConcept]
```

(Example)

This defines agt (agent) which is a case-relation. It is assumed not to have Attribute Definition. In Constraint, relation between either ?Do and ?NominalEntity or ?Action and ?NominalEntity must be satisfied.

```
{#agt Schema type='ElementalRelation';
  {#c Constraint at=#Event;
    {#a Arcs; [?Do-relationFrom->#agt-relationTo->?NominalEntity]}
    {#b Arcs; [?Action-relationFrom->#agt-relationTo->?NominalEntity]}}
  [#agt subRelationOf #caseRelation]}
```

(4) Complex Relation Type Schema

```
{#DefinitionLabel Schema type='complexRelation';
                        AttributeDefinition
                        StructureDefinitionOfComplexRelation
                        Constraint
                        Arc…}

Arc ::= [#DefinitionLabel subRelationOf #DefinitionLabelOfSuperiorConcept]
```

(Example)

The definition describes the complex relation based on the action "employ" in the relationship between "employer" and "employee", both of which are assumed to have no Attribute Definitions. First, there is a definition of `employer'. The higher-level concept of agentActor is assumed.

```
{#employer Schema type='complexRelation';
  {#s Structure type='relation';
    "employ#1
    "ToComplexRelation#2
    "FromComplexRelation#3
    "agt#4(type='relation')
    "obj#5(type='relation')
    {#a Arcs;
      [#1-relationFrom->#4-relationTo->#2]
      [#1-relationFrom->#5-relationTo->#3]}}
  {#c Constraint at=#Event;
    {#a Arcs; [?Human#6->#employer-relationTo->?Human#7]}}
  [#employer subRelationOf #agentActor]}
```

Next, the concept of `employee' is defined. The higher-level concept of objectActor is assumed.

```
{#employee Schema type='complexRelation';
  {#s Structure type='relation';
    "employ#1
    "ToComplexRelation#2
    "FromComplexRelation#3
    "agt#4(type='relation')
    "obj#5(type='relation')
    {#a Arcs;
      [#1-relationFrom->#4-relationTo->#3]
      [#1-relationFrom->#5-relationTo->#2]}}
  {#c Constraint at=#Event;
    {#a Arcs; [?Human#6->#employee-relationTo->?Human#7]}}
  [#employee subRelationOf #objectActor]}
```

(5)Attribute Type Schema

```
{#DefinitionLabel Schema type= 'attribute';
                        StructureDefinitionOfAttribute
                        Arc…}

Arc ::= [#DefinitionLabel subAttributeOf #DefinitionLabelOfSuperiorConcept]
```

[Example 1]

This defines an attribute-value pair for NominalEntity. For the attribute `realizationMode', a value set `realizationModeValue' and a default value 'def' are defined.

```
{#AttributeOfNominalEntity Schema type= 'attribute';
  {#s Structure type='attribute'
    realizationMode
    { 'generic' 'def' 'indef' }#realizationModeValue
    {#a1 Arcs;
      [#NominalEntity-attribute-> realizationMode]
      [realizationMode-value->#  realizationModeValue]
      [realizationMode-default->'def']}}}
  [#AttributeOfNominalEntity subAttributeOf #AttributeOfElementalEntity]}
```

[Example 2]

This defines AttributeOfEvent that is an attribute-value pair for Event.

```
{#AttributeOfEvent Schema type= 'attribute';
  {#s Structure type='attribute'
    {#a Arcs;
      [#Event-attribute->temporality]
      [temporality-value->{'past' 'present' 'future' 'permanent'}]
      [temporality-default->'present']
      [#Event-attribute->aspectuality]
      [aspectuality-value->{'begin' 'complete' 'continue' 'custom'
         'end' 'experience' 'progress' 'repeat' 'state' 'unspec'}]
      [aspectuality-default->'unspec']
      [#Event-attribute->aspectModify]
      [aspectModify-value->{'just' 'soon' 'yet' 'unspec'}]
      [aspectModify-default->'unspec']}}}
  [#AttributeOfEvent subAttributeOf #AttributeOfComplexEntity]}
```

Concept Definition Dictionary(CDD)

Concept definition dictionary (CDD) is one of the largest complex entities. It provides CDL with concept set. Concept definitions are described as realization of Schema of CDL.core and concept hierarchies are described as Arc of subconceptOf.

```
ConceptDefinitionDictionary::= {#DefinitionLabel ConceptDefinitionDictionary type='complexEntity';
        AttributeDefinition
        RealizationofConcept…
        Schema…
        Arc…}

{#CDD.xxx ConceptDefinitionDictionary Attribute-value…;
  {#Concept Concept}
  {#Entity Entity}
  {#ElementalEntity ElementalEntity}
  {#ComplexEntity ComplexEntity}
  {#Relation Relation}
  {#ElementalRelation ElementalRelation}
  {#ComplexRelation ComplexRelation}
  {#Attribute Attribute}
  RealizationOfElementalEntityTypeSchema…
  RealizationOfComplexEntityTypeSchema…
  RealizationOfElementalRelationTypeSchema…
  RealizationOfComplexRelationTypeSchema…
  RealizationOfAttributeTypeSchema…
  [#Entity subConceptOf #Concept]
  [#Relation subConceptOf #Concept]
  [#Attribute subConceptOf #Concept]
  [#ElementalEntity subConceptOf #Entity]
  [#ComplexEntity subConceptOf #Entity]
  [#ElementalRelation subConceptOf #Relation]
  [#ComplexRelation subConceptOf #Relation]
  LabelOfRealizationOfElementalEntityTypeSchema subEntityOf #ElementalEntity]…
  [LabelOf RealizationOfComplexEntityTypeSchema subEntityOf #ComplexEntity]…
  [LabelOf RealizationOfElementalRelationTypeSchema
            subRelationOf #ElementalRelation]…
  [LabelOf RealizationOfComplexRelationTypeSchema
            subRelationOf #ComplexRelation]…
  [LabelOf RealizationOfAttributeTypeSchema subAttributeOf #Attribute]…
  [Label-e1 subEntityOf Label-e2]…
  [Label-r1 subRelationOf Label-r2]…
  [Label-a1 subAttributeOf Label-a2]…}
```

Inheritance Mechanism for Concept Hierarchy

A concept definition consists of content described as realization of the schema and content inherited from higher-level concepts linked in hierarchical relations of concepts. What higher-level concept definitions are inherited and how, as well as how definition content specifies realization content, is specified with these inheritance mechanisms.

First is the single inheritance mechanism from a higher-level concept. This inheritance mechanism is specified by the inheritance mechanism for attributes, structure and constraints.

1. Inheritance mechanism for attributes: Inheritance mechanism for attribute types schema, that is, the inheritance mechanism for structure definitions of attributes.
2. Inheritance mechanism for structures: A structure definition consists of constituent elements of the structure, enumerations of the elements and constraints between the elements. Among the elements, those that have replaced elements in higher-level concepts and those that have been added newly will take priority, and all others will inherit elements of higher-level concepts. Constraints follow the inheritance mechanism for constraint definitions.
3. Inheritance mechanism for constraints: Constraints are defined by inheritance mechanism based on entity arcs as units. In other words, constraints that have replaced entity arcs in a higher-level concept with identifiers for entity arc realization labels, as well as those that have been added newly, will take priority. All others will inherit entity arcs of higher-level concepts.

Next is a multiple inheritance mechanism for multiple higher-level concepts. Multiple inheritances require inheritance of multiple properties as logical sum. In other words, elements constituting structure definition will be inherited as sum set, and constraints will be inherited in enumeration form as sum set of entity arc.

Basic Operations on CDD

Basic operations on CDD and schema are defined. Here, operations are described in predicate form. However, these operations can be described as complex entities. SchemaPattern is realization schema including variables. With this SchemaPattern description capability, additional operations for schema search, verification, decomposition and synthesis are made available. The basic operations are as follows.

Create CDD:

CreateCDD(CDDDeclaration, NewCDD)

Merge CDD:

MergeCDD(CDD1, CDD2, MergedCDD)

Deleate CDD:

DeleteCDD(CDD)

Add schema:

AddSchema(CDD, RealizedSchema, UpdatedCDD)

Delete schema:

DeleteSchema(CDD, SchemaDefLabel, UpdatedCDD)

Search schema:

SearchSchema(CDD, SchemaPattern, SchemaDefLabels)

Verify schema:

VerifySchema(CDD, RealizedSchemaOrSchemaPattern)

Decompose schema:

DecomposeSchema(RealizedSchema, SchemaPattern)

Compose schema:

ComposeSchema(SchemaPattern, RealizedSchema)

4.2 CDL.nl

The CDL.nl is an artificial language designed to describe every information written in natural languages. CDL.nl is a language independent from any natural languages and provides the frame of language dependent concept description language for natural language so called CDL language family such as CDL.eng (CDL for English), CDL.jpn (CDL for Japanese), CDL.chn (CDL for Chinese), etc.

For that purpose CDL.nl has a top-ontology to be common to every CDL language family even they are language dependent. This top ontology provides a semantic background for CDL language family and enables computers to process information written in CDL language family semantically.

CDL.nl expresses a meaning of a text written in natural languages in the form of hyper semantic graph. In CDL.nl hyper graph, a node represent a concept expressed as a document, a sentence, a clause, a phrase, a word or a morpheme and an arc represent a relation between concepts.

CDL.nl describes objectivity of a sentence and subjectivity of a sentence separately. Objectivity is described set of concept pair with relation, and subjectivity is described by attributes attached to concepts.

The CDL (Concept Description Language) is a language proposed by ISeC for Semantic Computing (SeC) R&D. CDL is fundamental language within SeC (Semantic Computing) framework.

CDL describes semantic/conceptual structure of contents (resources) by CDL Model & Syntax. It provide basic data structure common to all CDL.*

The following is the basic data structure of CDL.core.

Basic data structure

`{' `}'	Concept defined in the CDL
`<' `>'	Concept which is imagined by the symbol.

```
{ Instance Label Concept Label Attribute Value Pair… :
                 Concept…
                 Arch… }

Arch:= [node1   node2   node3]
```

The following is a example of CDL.nl expression for the sentence `I recieved a report that a computer was purchased yesterday.'.

```
{#A event:
  {#B event:
    <#11:yesterday>
    <#12:computer>
    <#13:purchased>
    [#13 tim #11] [#13 obj #12]
  }
  <#2:report>
  <#3:received>
  <#4:I>
  [#2 cnt #B]  [#3 obj #2]  [#3 rec #4]
}
```

4.1.1 CDD.nl

A language is a system consists of syntax and vocabulary. Each word of a language expresses a concept or a relation among sentential elements. A concept has a semantic background which is clear to users of a language.

It is necessary for CDL.nl to have vocabulary with semantic background to be a language to describe something. The Concept Definition Dictionary for Natural Languages (CDD.nl) provides the vocabulary of CDL.nl and semantic background of each concept.

The top ontology of CDL.nl is developed based on the UNLKB (Knowledge Base of Universal Networking Language). We employ UWs (Universal Words of UNL) as vocabulary of CDL.nl. The UNL including UNLKB and UWs is developed under United Nations University /

Institutes of Advanced Sturdy from 1996, and research and development were transferred to the UNDL Foundation in 2001. Whole UW set is defined at http://www.undl.org/unlsys/uw/unlkb.htm.

4.1.2 Simple Syntax

The simple syntax for CDL.nl is introduced for readability, writeability and understandability. It is introduced to describe meaning of texts in a compact and simple way. In this document, the simple syntax is used for showing examples of CDL.nl expressions (CDL.nl graphs) together with those of CDL.core.

Simple Syntax of a CDL.nl graph

A CDL.nl graph expresses a (compound) concept, a compound concept is composed by a set of directed binary relations, and each binary relation is made up of a relation and two concepts. Concepts that construct a CDL.nl graph can also be CDL.nl graphs. Basic components of a CDL.nl graph are names of concepts, relations and attributes. For names of concepts see 3 Entities, relations are described in 4 Relations, and attributes are described in 5 Attributes.

Simple Syntax

The simple syntax is specified here by means of extended BNF. In the description of simple syntax, terminals are quoted, and non-terminals are bold and not quoted. Alternatives are separated by vertical bars (|). Components that can be omitted are enclosed with square brackets ([…]). Components that can occur any times are enclosed with braces ({…}). Repetition of a component is followed by three dots (…). White spaces should be ignored (except the case mentioned below in note 2).

Concept	=:: { ConceptLabel | CompoundConcept } ':' ConceptID ':' Attributes
CompoundConcept	=:: '{' Concept1 Relation Concept2 [[','] Relation Concept2] … [';' Concept1 Relation Concept2] … '}'
Relation	=:: { '<' | '>' } RelationLabel
RelationLabel	=:: 'agt' | 'and' | …
ConceptLabel	=:: a literal expression (must be in double quotes) | a label of concept defined in CDD.nl | an UW
ConceptID	=:: any two characters of '0' – '9', 'a' – 'z', and 'A' – 'Z'
Attributes	=:: Attribute ['.' Attribute] …
Attribute	=:: 'entry' | 'topic' | 'past' | …

Notes:

1. When ConceptLabel is a literal expression, ConceptID can be omitted.
2. A space (or more) is necessary between Relation and Concept (1 or 2) in each binary expression of CompoundConcept.
3. Direction '>' of Relation is the default.
4. Repetition of a concept (ConceptLabel or CompoundConcept) is available by simply indicating the ConceptID only of the Concept, as in `:0A'. The concept must appear somewhere within the same CDL.nl graph and is provided a unique ConceptID. Attributes can be omitted in a repetition description.
5. The main concept of a compound concept must be given the attribute 'entry'.
6. Language type (attribute) of ConceptLabel must be indicated, except for the names of entities defined in CDD.nl. A language type can be indicated in the entity (concept) description as the value of attribute lang as lang="en", and/or in Attributes field of the simple syntax. Language types are 'en', 'ja', 'uw', 'zh', etc. For details see section 5.8 Describing language types of concept names.

Meaning of the simple syntax

This section describes the functions of delimiters and uses of the simple syntax for various meanings. Delimiters used in the simple syntax are braces ({…}), commas (,), semicolons (;) and dots (.). The delimiters describe the structure of a CDL.nl expression (graph); Components provide meaningful information of the CDL.nl expression.

1. { and } define a compound concept. This means that any compound concept must start with '{' and end with '}'.
   In each compound concept, a concept must be attached with attribute 'entry' and the concept is the main in the compound concept.
   For example { red:01:entry >aoj apple:02 } means `an apple is red', whereas { red:01 >aoj apple:02:entry } means `a red apple'.
2. A comma (,) introduces a pair of a relation and a concept, to connect the relation to the concept on the opposite side of the previous relation.
   For example, `I came yesterday' can be expressed as:

   ```
   { come:01:entry.past  >agt  I:02 ,
                         >tim  yesterday:03 }
   ```

   This CDL.nl expression means that two relation 'agt' and 'tim' emerge from 'come:01:entry.past'.
3. A chain of concepts can be described simply by repeating relations and concepts.
   For example, `the vase bought yesterday is broken' can be expressed as:

   ```
   { yesterday:01  <tim  buy:02:past  >obj  vase:03:def  <aoj  broken:04:entry }
   ```

4. A semicolon (;) introduces a new binary relation in a compound concept.
   Using semicolons, for example, `the vase bought yesterday is broken' can be expressed as:

```
{ buy:01:past  >tim  yesterday:02;
  buy:01:past  >obj  vase:03:def;
  broken:04:entry  >aoj  vase:03:def
}
```

5. Repetition of a concept within a sentence (CDL.nl graph) can be done by simply indicating the ID of the concept.
   Using the function of repetition of concept, for example, `the vase bought yesterday is broken' can be expressed as:

```
{ buy:01:past  >tim  yesterday:02 ;
  :01          >obj   vase:03:def ;
  broken:04:entry  >aoj  :03
}
```

   ':01' in the second binary relation and ':03' in the third are repetition descriptions, they are meant the concepts of 'buy:01:past' and 'vase:03:def' respectively.
   For this purpose, IDs of concepts no matter what types (a single or compound concept) the concepts are or where they are (inside or outside a compound concept) must be given uniquely within a CDL.nl graph. The uniqueness of ID guarantees the repetition description meaningful. Repetition description is also available about a concept inside a compound concept from the outside, and vice versa.
6. A compound concept should be also given a unique ID and can be referred to using the ID.
   For example, `I like sweet oranges and apples' can be expressed as:

```
{ { orange:01  <and  apple:02:entry }:03  <aoj  sweet:04 ;
  like:05:entry  >obj  :03 , >agt  I:06
}
```

   ':03' placed in the second binary relation means the compound concept { orange:01 <and apple:02:entry }.
7. A dot (.) separates two attributes.

Examples

The sentence below enclosed with {org} and {/org} is the original text from the article `Tsunamis: causes, consequences, prediction and response' from EOLSS. <cdl>…</cdl> shows the CDL.nl expression in simple syntax.

In the example of CDL.nl expression, 'uw' was attached at the end to indicate that all ConceptLabels of the CDL.nl expression are UWs (Universal Words) of UNL.

```
<cdl:>
{
present:0D:entry  >agt   article:05:topic;
:0D  >obj  essential:0Q:def.pl;
:0Q  >mod  event:1E:indef;
induce(icl>cause:1X:may  >obj  :1E;
:1E  > mod  tsunami:16;
:1X  >agt
        {
        decomposition:5P:entry.def  >or  fall:5D;
        :5P  >plc  shelf:6R:def;
        :5P  >obj  hydrate:6C:indef;
        hydrate:6C:indef  >mod  gas:68;
        :5D  >or  disturbance:4O;
        cosmic:51  >aoj  :5D;
        :5D  >obj  body:58;
        :4O  >or  eruption:42;
        :4O  >mod  atmospheric:4C;
        :42  >or  landslide:38:indef;
        :42  >mod  volcan3U;
        :3U  >mod  underwater:3J;
        :38  >or  earthquake:2K:indef;
        :38  >mod  submarine:2Y;
        :2K  >mod  bottom:2D;
        }:01;
:05  >mod  this:00;
}::uw
</cdl>
```

Cross-reference of concepts

Cross-reference between concepts is described using relation 'ref'. 'ref' links a concept that is considered to be a referent, with another concept that is considered to be the referee. The syntax is the following:

```
ReferentConcept  >ref  RefereeConcept
```

or

```
RefereeConcept  <ref  ReferentConcept
```

Where,

- ReferentConcept is a concept that refers to RefereeConcept, and
- RefereeConcept is a concept that ReferentConcept refers to.

Reference description must be described in the CDL.nl where ReferentConcept is included. If the referee is in the same CDL.nl graph, the Concept of the referee is simply used in the reference description. If a concept in other sentence (CDL.nl graph) is referred to, the SentenceID (and the ParagraphID) must be attached. A sentence or a paragraph can be referred to using SentenceID and ParagraphID respectively.

Simple syntax of ReferentConcept is the following:

ReferentConcept	=:: Concept

Simple syntax of RefereeConcept is the following:

RefereeConcept	=:: ParagraphID | SentenceID | Concept [':' SentenceID [':' ParagraphID]]

where,

ParagraphID	=:: { 'P' } ParagraphNumber
SentenceID	=:: { 'S' } SentenceNumber
ParagraphNumber	=:: a number expressed by '0' – '9'
SentenceNumber	=:: a number expressed by '0' – '9'

Syntax of a CDL.nl document

CDL.nl document	=:: '<cdl:' DocumentInfomation '>' { '<P:' ParagraphNumber '>' { '<S:' SentenceNumber '>' SentenceConcept '</S>' }… '</P>' }… '</cdl>'
SentenceConcept	=:: Concept

Comments:

'<S:number>' and '</S>' should be possible to omit if there is a way to describe SentenceNumber instead, however.

Concepts

A concept of CDL.nl can be a class concept, an instance, a single concept or a compound concept, and is allowed to be expressed by a literal expression of a natural language such as a word, a phrase or a sentence, a name of an class concept defined in CDD,nl such as `nominal concept', `predicative concept', etc.. Whole set of UWs of UNL are defined at www.undl.org/unlsys/uw/unlkb.htm. In this document, the term of concept label is used to express an expression of concept of CDL.nl.

Concept labels can be modified (restricted) by attributes in order to express various concepts. For example, `book':01:def.en means some specific book that mentioned before, where `book' expresses a class concept.

Concepts are connected to each other by relations to make up CDL.nl graphs of compound concepts. A CDL.nl graph can also be connected with other (compound) concepts.

5. CWL.rdf

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery.

RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values.

RDF provides possibility to define graphs which are mostly meant for representation of key properties of information resources. This structure can also be used for representation od other kind of information, such as semantic structure of the sentence. CWL.rdf is using RDF sinthax to represent conceptual network (graph) of the sentence.

5.1 RDF Graph Structure

CWL.rdf introduces 3 basic resource types:

1. http://www.undl.org/unl/uw
2. http://www.undl.org/unl/relation
3. http://www.undl.org/unl/sentence

5.1.1 http://www.undl.org/unl/uw resource type

This resource type stands for the definition of the concepts (nodes) present in the graph of the sentence.

Below is an example of a concept definition using CWL.rdf:

```
<rdf:Description rdf:about='http://www.undl.org/unl/uw#s1-2D'>
    <rdf:type rdf:resource='http://www.undl.org/unl/uw'/>
    <unl:headWord>scientist</unl:headWord>
    <unl:id>2D</unl:id>
    <unl:entityReferenceString><![CDATA[(icl>scholar)]]></unl:entityReferenceString>
    <unl:hasAttribute rdf:resource='http://www.undl.org/unl/attribute#@pl'/>
</rdf:Description>
```

It consists of following properties:

1. **unl:headword** – A string indicating the concept in English.
2. **unl:id** – an identifier that guarantee uniqueness of the concept in the sentence scope.
3. **unl:entityReferenceString** – a string value indicating the references to the UNL Knowledge Base
4. **unl:hasAttribute** – a reference to predefined UNL attribute

5.1.2 http://www.undl.org/unl/relation resource type

This resource type stands for the definition of the relations (arches) present in the graph of the sentence.

Below is an example of a relation definition using CWL.rdf:

```
<rdf:Description rdf:about='http://www.undl.org/unl/relation#s1-ss-r4-and'>
    <rdf:type rdf:resource='http://www.undl.org/unl/relation'/>
    <unl:relationType rdf:resource='http://www.undl.org/unl/relation-type#and'/>
    <unl:sourceEntity rdf:resource='http://www.undl.org/unl/uw#s1-5S'/>
    <unl:targetEntity rdf:resource='http://www.undl.org/unl/uw#s1-33'/>
</rdf:Description>
```

It consists of following properties:

1. **unl:relationType** – a reference to a predefined relation type.
2. **unl:sourceEntity** – a reference to a concept resource, which correspond to the source node of the arch arrow.
3. **unl:targetEntity** – a reference to a concept resource, which correspond to the target node of the arch arrow.

5.1.3 http://www.undl.org/unl/sentence resource type

This resource type stands for the definition of the sentence.

Below is an example of a sentence definition using CWL.rdf:

```
<rdf:Description rdf:about='http://www.undl.org/sentence#s1'>
    <rdf:type rdf:resource='http://www.undl.org/unl/sentence'/>
    <unl:hasRelation rdf:resource='http://www.undl.org/unl/relation#s1-ss-r0-obj'/>
    <unl:hasRelation rdf:resource='http://www.undl.org/unl/relation#s1-ss-r1-man'/>
    <unl:hasRelation rdf:resource='http://www.undl.org/unl/relation#s1-ss-r2-tim'/>
    <unl:hasRelation rdf:resource='http://www.undl.org/unl/relation#s1-ss-r3-aoj'/>
</rdf:Description>
```

It consists of following properties:

- **unl:hasRelation** – a reference to a relation.

6. CWL platform

As we do not have conversion system between natural languages and CWL, we use the UNL system for input CWL and output what is expressed in CWL in natural languages. In that sense, the UNL system is the infrastructure of CWL platform.

The CWL platform consists of CWL Editor, CWL converter and the UNL System consisting of Enconverter (which convert natural languages into UNL), Deconverter (which convert UNL into natural languages).

Since compatibility of the three types of expression of CWL.unl, CWL.cdl, CWL.rdf, The UNL system together with conversion system allow people to make web pages in CWL.unl, CWL.cdl, CWL.rdf and also allow people to see those web pages in their mother tongues.

Beside the UNL system, we provided conversion system (CWL converter) among CWL.unl, CWL.cdl and CWL.rdf. And also we provide necessary vocabulary for CWL based on the UNLKB which provides semantic background of Universal Words of UNL. Knowledge on words (CWL ontology) of each language is stored in UNLKB (CWL.unl), CDD.nl (CWL.cdl) and OWL (CWL.rdf).

6.1 CWL Editor

The CWL editor is used to input CWL using natural languages and display CWL in natural languages. The CWL Editor has the following functions.

1. Narural languge conversion into CWL by Word Selection Based Analysis.
2. HTML file handler
3. Graphcal representation of CWL
4. Conversion among CWL graph, CWL.unl, CWL.cdl, CWL.rdf
5. Natural language generation from CWL

The CWL Editor Is a web based IDE, which provides basic functionality for the modification and validation of the inserted code. The following technologies and Programming Languages were used:

1. **Java** – serverside processing, parsing.
2. **JSP/Servlet** – Web rendering
3. **JavaScript/DHTML** – clientside processing and rendering
4. **Ajax** – server inquiries and data submition.
5. **VML(Vector Markup Language)** – graphical rendering of the parsed object graph.

6.2 CWL System

The UNL System consists of three major components: language resources, software for processing the language resources, and tools and systems for maintaining and operating the language processing software or language resources. Language resources are divided into language dependent part and language independent part. Linguistic knowledge on concepts that universal to every language is considered language independent and to be stored in the common database UNLKB. Language dependent resources like word dictionaries and rules, as well as the software for language processing, are stored in each language server. Language servers are connected in the Internet through UNL. Supporting tools for producing UNL documents can be used in a local PC. Such supporting tools function with consulting language servers through the Internet. Verification of UNL documents can be carried out through the Internet or in a local PC. The tool (UW Gate) for search or maintenance of the common database UNLKB functions through the Internet.

The structure of the UNL System

In figure 4, the highlited parts show the components of the UNL System. White parts show applications and their UNL database based on the UNL System.

Each component of figure 4 is explained below, from the upper right:

- **UNLKCIC** stores information of Key Concept in Context (KCIC) about UNL documents. The KCIC is made for every binary relation of UNL documents. This information is used when searching for related UNL expressions of a UNL expression. Through UNLKCC, each UWs in the UNLKB is linked to the instance of it, then world knowledge on UNL is described as relations between other UWs.
- **Concept Definitions** are the collection of definitions of UWs. Definitions are provided by sentences and are expressed in UNL (document). Definitions (semantics/meaning) of concepts provide the knowledge of the concepts in connection with other concepts that can specify the concepts. This knowledge is indispensable for reasoning in information retrieval, etc.

Figure 4. Structure of the UNL System

- **UNL Documents** mean the documents in which UNL expression is described for each sentence of natural language. A UNL document can be made of a plain text file or an UNL-embedded html file. A UNL Document base is a collection of UNL Document files. UNL documents are for the purpose to provide information in UNL.
- The **UNLKB** is explained in previous section 2.1.
- **UW Dictionary** provides the interface between UWs and words of natural languages.
- The **UNL Verifier** verifies whether a UNL expression is correct syntactically, lexically and semantically. The syntax check of a UNL expression is carried out against the UNL Specifications. In lexical check, whether all UWs of a UNL expression are defined in the UNLKB are checked. In semantic check, whether each binary relation of a UNL expression is defined as possible is certified with consulting the UNL KB.
- UNL **Language Servers (LSs)** are located in the Internet to carry out the conversions between natural languages and UNL expressions. Each LS contains an EnConverter and a DeConverter of a language. EnConverter converts natural language sentences to UNL expressions. DeConverter convert UNL expressions to natural language sentences.
- The **UW Gate** is tool for people to access the UNLKB and the UW dictionary through the Internet. The use of the UW gate is authorized to the UNL Society members in three levels. First, every member of the UNL Society is allowed to search the UNLKB and UW dictionary for necessary information. Second, members that belong to a particular language group are allowed to modify the links between the language and UWs. Third, members that have mastered how to define UWs are allowed to register new UWs to the UNLKB.
- The **UNL Proxy Server** functions to communicate with language servers. It functions as a filter to check whether a web page that a user required is written in UNL or not. If UNL expressions are included in the web page, it communicates with an appropriate language server in the Internet for deconverting the UNL expressions into desired language sentences and provides the Internet browser with the results to display.
- The **UNL Editor** is a tool helps to produce UNL documents. It includes an EnConverter and a DeConverter. Each of them can be selected according to language. EnConverter converts natural language sentence into UNL expressions. Whereas DeConverter provides

generated results as feedback for checking the correctness of UNL expressions.

- The **UNL Explorer** provides the basis of knowledge infrastructure. The UNL Explorer manages UNLKB, UNLKCIC and UNL Documents provide the knowledge on UWs. The UNL Explorer has two types of function. For human, the UNL Explorer allows users or developers to view or to develop the UNL Knowledge System such as the UNL Encyclopaedia. For computers, it provides information or knowledge on UWs.
- The UNL Explorer uses UNLKB for navigating information stored in UNL database. It has two windows: the hierarchy of UWs (UW System) of UNLKB is shown in the left window. UWs of the UNLKB are keys for information stored in UNL database. Information on UWs is shown in the right window through navigation through the UW System. Information on UWs is described in UNL documents. All UWs used in the UNL documents are included in the left window of UNLKB and are keys for further information.
- The UNL Explorer allows users to search for information using UWs or words of natural languages. It shows the information in UNL or a desired natural language by accessing UNL Language Servers. It also provides functions for developers to add information to or modify information of the UNL database in their native languages.
- Information about a UW is stored in a file. Location of the file is linked with the UW. This architecture of the UNL Knowledge System allows its development to be carried out by a wide range of developers from different languages and cultures. Such a database can provide a wealth of up-to-date information on various aspects of information and knowledge from all over the world.

Mechanism of conversion of UNL expressions

Figure 5. Mechanism of conversion

Figure 5 shows the mechanism how a UNL document is made and how a UNL document is converted into natural languages in the UNL system. The EnConverter and DeConverter are the core software in the UNL system. The EnConverter converts natural language sentences into UNL expressions. The Universal Parser (UP) is a specialized version of th3.e EnConverter. It generates UNL expressions from annotated sentences using the UW dictionary only. All UNL expressions are verified by the UNL verifier. The DeConverter converts UNL expressions to natural language sentences.

6.3 CWL Converter (Conversion among CWL.cdl, CWL.unl, CWL.rdf)

The CWL Conversion Framework serves as a tool for conversions among CWL.unl, CWL.cdl and CWL.rdf. It is implemented as a web application written using Java/JSP, DHTML/Ajax and VML.

6.3.1 Application Flow

The basic flow of the conversion considers:

1. Parsing of input document
2. Building of object graph, where nodes correspond to entities and arches correspond to relations/predicates.
3. Generation of various views (CWL graph, CWL.unl, CWL.cdl, CWL.rdf)

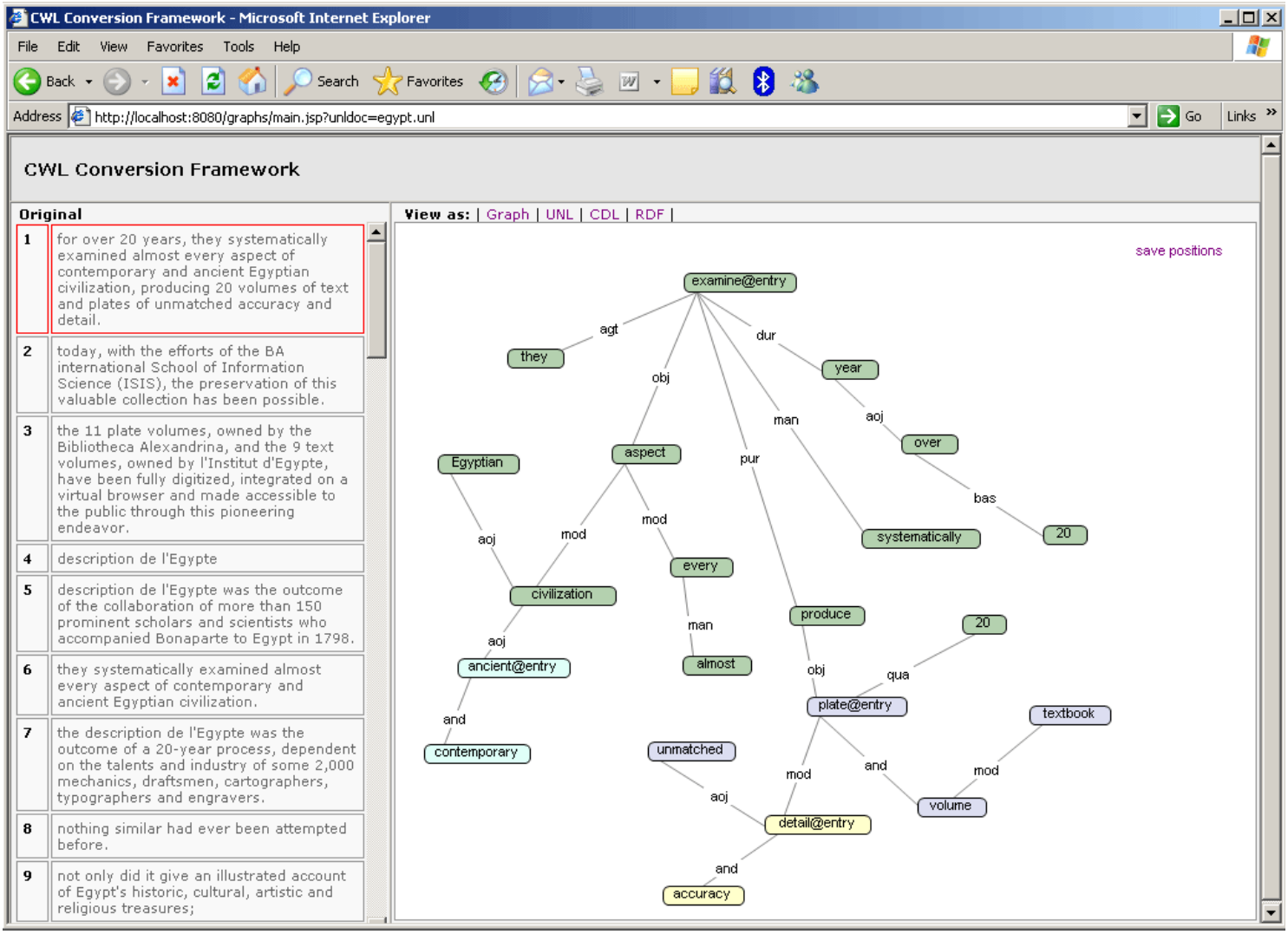
All basic functionality is implemented as a Java library which can be reused for various applications.

6.3.2 View Rendering

Object graphs can be rendered in different ways but in general they reflect the same relational network from the input document.

CWL (Graph) View

The default is a graphical view where the sentence from natural language is presented as a graph (see Picture 1). Nodes of the graph contain Universal Words from UNL KB corresponding to the entities building the natural language sentence.



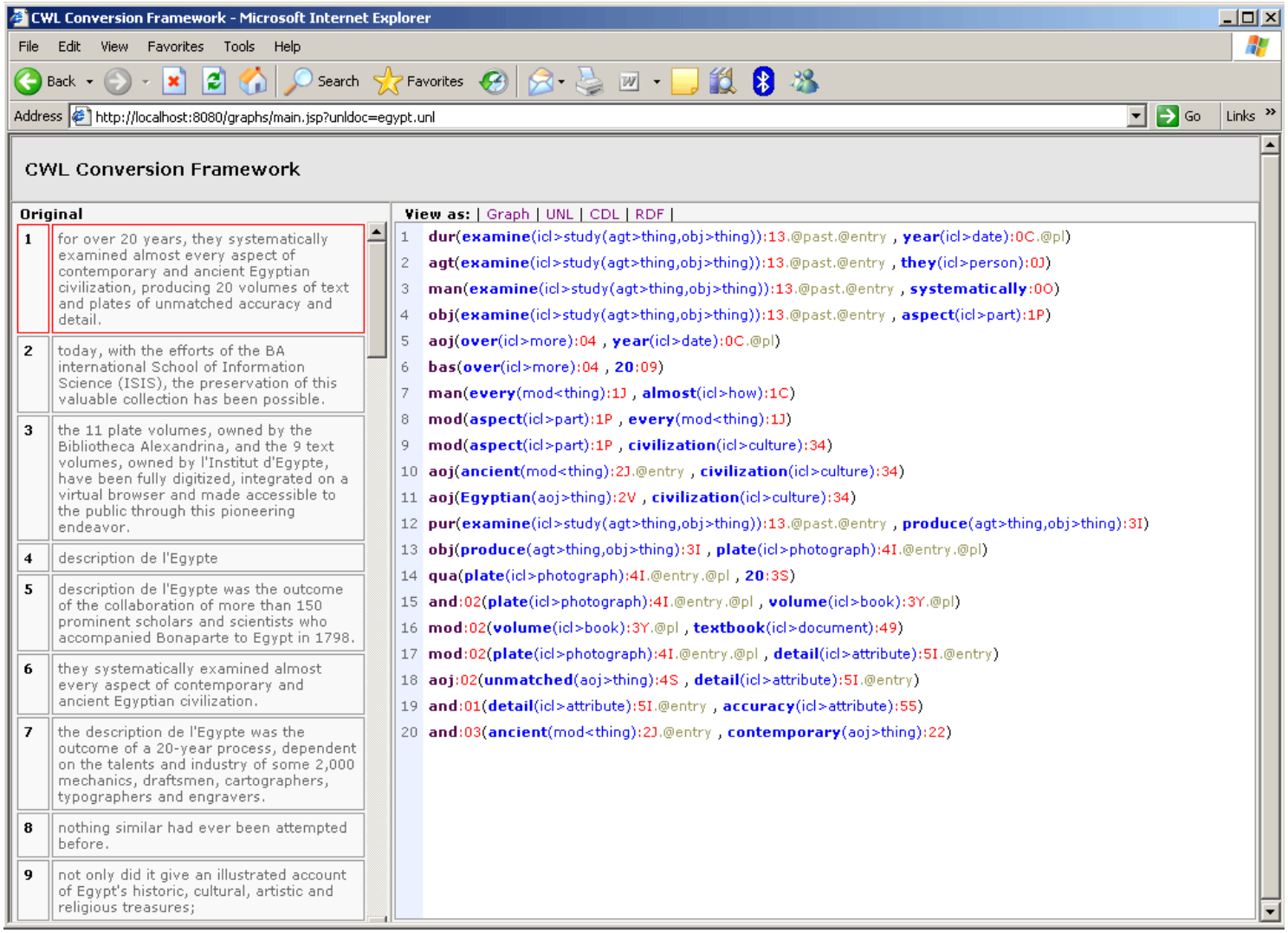
Picture 1: CWL (Graph) View

Arches of the graph correspond to the relations among the entities. The direction of the relation can be seen by clicking on any node. In that case you will see incoming relations colored as red lines and outgoing relations will remain black. Full versions of Universal Words can be seen by double-clicking on the nodes, otherwise only headwords are shown by default. The rendering is done using VML (Vector Markup Language). Nodes of the graph are dragable, thus each user can customize the look for his convenience. After dragging the nodes, the overall positions of the nodes can be saved by clicking corresponding link in upper right corner of rendering area.

Sub sentences of the sentence are rendered using different colors for nodes. All sub sentences have root entry marked with `@entry' string after headword. Same refers to the root entry of the sentence itself.

CWL.unl View

The UNL View renders the graph as a sequence of UNL relations (see Picture 2). The code is highlighted accordingly. The sequence of the relations considers main sentence relations first and sub sentence relations afterwards.

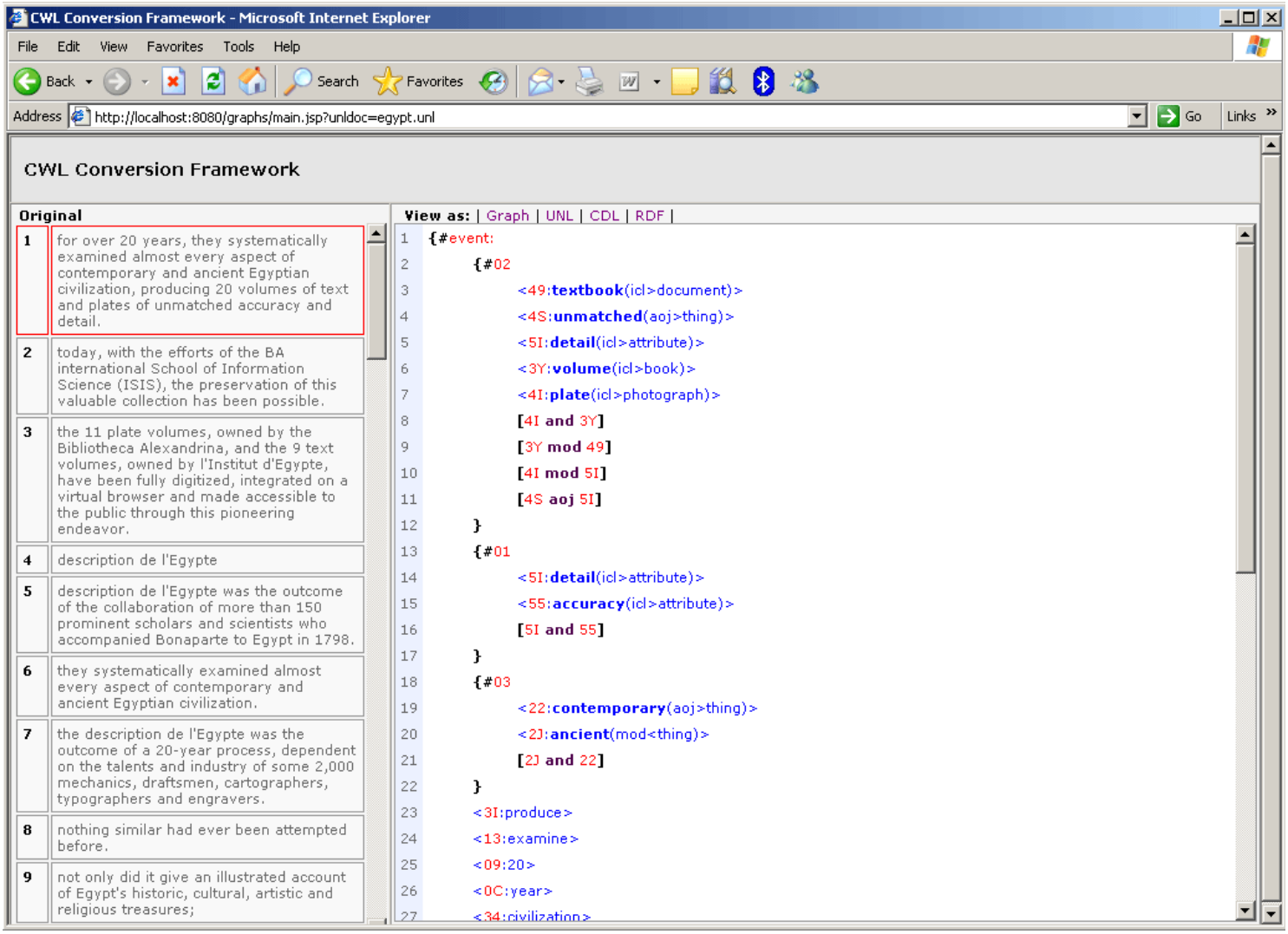


Picture 2: UNL View

CWL.cdl View

The CDL View renders the graph as sequences of entries and relations (see Picture 3).

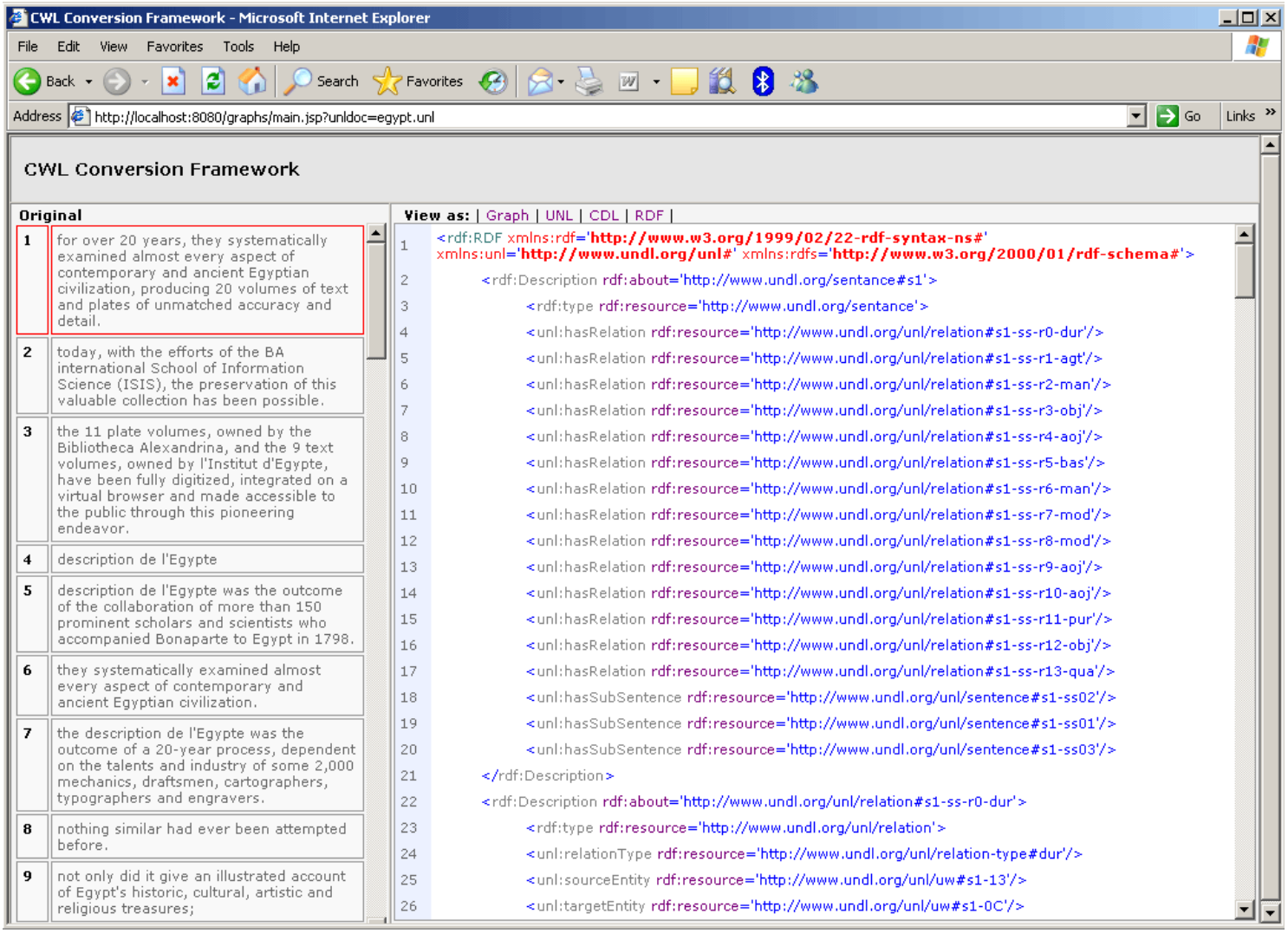
At first the entity sequences appear where UWs from UNL KB are being used. After which there relations among listed entities are rendered. Similar structure is being used for sub sentences. The code is highlighted accordingly.



Picture 3: UNL View

CWL.rdf View

The RDF View renders the graph as an RDF document where Sentences(sub sentences), Relations and Entities (Universal Words) are defined as objects using `<rdf:description>' tags (see Picture 4). Corresponding namespaces and unique URIs are being generated and referenced. The code is highlighted accordingly.



Picture 4: RDF View

Many applications that process multimedia assets make use of some form of metadata that describe the multimedia content. The goals of this document are to explain the advantages of using Semantic Web languages and technologies for the creation, storage, manipulation, interchange and processing of image metadata. In addition, it provides guidelines for Semantic Web-based image annotation, illustrated by use cases. Relevant RDF and OWL vocabularies are discussed, along with a short overview of publicly available tools.

7. CWL and Web

7.1 How CWL is useful for Web Community

There are several roles of CWL which contribute to more intelligent web handling. The following is some examples of these directions.

1. **intelligent web searching**
   The future web search mechanism will be based on the semantics of web contents, and in that case they should be written in more intelligent and well defined. In order to do it, it is desirable for users to make use of some kind of intelligent description language. CDL might be one of this kind of web description languages.
2. **intelligent ontology development**
   Semantic web is based on the elaborated definition of ontology, and OWL is a language of the ontology based on the RDF. It is possible to make more fundamental definition through the use of CDL, a base language for human concept definition.
   XML, a common base language for Web technology, and there are several kinds of XML extension for specific application fields like XBRL for business report language for financial report development. In order to really use of these kinds of XML extension, it is necessary to prepare the taxonomy for each of the application fields. This mechanism is very similar to the ontology development in semantic web technology.
   CWL starts from CDL, a common base language for human concept, and it is very useful to use CWL or its derivatives to develop elaborated ontology and taxonomy.
3. **web pages translation**
   Web pages and contents are written in natural languages like Japanese, English and so on. There are several translation engine for web pages from one natural language into another. CDL.unl is used to translate UN documents into several natural languages, and CWL.unl is also useful like CDL.unl.

Reference

1. Hiroshi Uchida, Meiying Zhu, Tarcisio G. Della Senta: UNIVERSAL NETWORKING LANGUAGE, UNDL Foundation, 2005

Acknowledgments

The editor would like to thank to Prof. Ishizuka and other members of the XG for their corporation and feedback for the development of CWL.
