IETF media feature registration WG
INTERNET-DRAFT
Category: Work-in-progress

Graham Klyne
Content Technologies/5GM

September 1998
Expires: March 1999

A syntax for describing media feature sets

<draft-ietf-conneg-feature-syntax-00.txt>

Status of this memo

[[INTENDED STATUS:
This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.]]

Copyright Notice

Abstract

A number of Internet application protocols have a need to provide content negotiation for the resources with which they interact [1]. A framework for such negotiation is described in [2]. Part of this framework is a way to describe the range of media features which can be handled by the sender, recipient or document transmission format of a message. A format for a vocabulary of individual media features and procedures for registering media features are presented in [3].

This document introduces and describes a syntax that can be used to define feature sets which are formed from combinations and relations involving individual media features. Such feature sets are used to describe the media feature handling capabilities of message senders, recipients and file formats.

This document also outlines an algorithm for feature set matching.

Table of contents

1. Introduction

A number of Internet application protocols have a need to provide content negotiation for the resources with which they interact [1]. A framework for such negotiation is described in [2]. A part of this framework is a way to describe the range of media features which can be handled by the sender, recipient or document transmission format of a message.

Descriptions of media feature capabilities need to be based upon some underlying vocabulary of individual media features. A format for such a vocabulary and procedures for registering media features within this vocabulary are presented in [3].

This document defines a syntax that can be used to describe feature sets which are formed from combinations and relations involving individual media features. Such feature sets are used to describe the media handling capabilities of message senders, recipients and file formats.

This document also outlines an algorithm for feature set matching.

The feature set syntax is built upon the principle of using feature set predicates as "mathematical relations" which define constraints on feature handling capabilities. This allows that the same form of feature set expression can be used to describe sender, receiver and file format capabilities. This has been loosely modelled on the way that relational databases use Boolean expresions to describe a set of result values, and a syntax that is based upon LDAP search filters.

1.1 Structure of this document

The main part of this memo addresses the following main areas:

Section 2 introduces and references some terms which are used with special meaning.

Section 3 introduces the concept of describing media handling capabilities as combinations of possible media features, and the idea of using Boolean expressions to express such combinations.

Section 4 contains a description of a syntax for describing feature sets based on the previously-introduced idea of Boolean expressions used to describe media feature combinations.

Section 5 discusses some feature set description processing issues, including a description of an algorithm for feature set matching.

1.2 Document terminology and conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

> NOTE: Comments like this provide additional nonessential information about the rationale behind this document. Such information is not needed for building a conformant implementation, but may help those who wish to understand the design in greater depth.

1.3 Discussion of this document

Discussion of this document should take place on the content negotiation and media feature registration mailing list hosted by the Internet Mail Consortium (IMC):

Please send comments regarding this document to:

ietf-medfree@imc.org

To subscribe to this list, send a message with the body 'subscribe' to "ietf-medfree-request@imc.org".

To see what has gone on before you subscribed, please see the mailing list archive at:

http://www.imc.org/ietf-medfree/

1.4 Amendment history

00a 28-Sep-1998 This memo created to contain a description of the syntax-related features from a previous.draft "An algebra for describing media feature sets". Theoretical background material is replaced by a more practically oriented introduction to the concepts, and references to ASN.1 representation have been removed.

Revision history of "An algebra for describing media feature sets":

00a 11-Mar-1998 Document initially created.

01a 05-May-1998 Mainly-editorial revision of sections describing the feature types and algebra. Added section on indicating preferences. Added section describing feature predicate syntax. Added to security considerations (based on fax negotiation scenarios draft).

01b 25-Jun-1998 New Internet draft boilerplate in 'status' preface. Review and rationalization of sections on feature combinations. Added numeric expressions, named predicates and auxiliary predicates as options in the syntax. Added examples of text string predicate representation.

02a 08-Jul-1998 Added chapter on protocol processing considerations, and in particular outlined an algorithm for feature set matching. Added restrictions to the form of arithmetic expression to allow deterministic feature set matching.

03a 27-Jul-1998 Simplified feature set handling by removing options for expressions on the RHS of feature comparison expressions. Syntax elements have been added as placeholders for possible future extensions in this area; examples have been adjusted accordingly, and the feature set matching algorithm greatly simplified. Add simple unit designations.

1.5 Unfinished business

- Discuss determination of qvalues in the feature set matching algorithm.

- Use of unknown data types for feature values (section 5.3)

- Add worked example and source code for feature matching implementation.

2. Content feature terminology and definitions

Feature Collection
: is a collection of different media features and associated values. This might be viewed as describing a specific rendering of a specific instance of a document or resource by a specific recipient.

Feature Set
: is a set of zero, one or more feature collections.

Feature set predicate
: A function of an arbitrary feature collection value which returns a Boolean result. A TRUE result is taken to mean that the corresponding feature collection belongs to some set of media feature handling capabilities defined by this predicate.

Other terms used in this draft are defined in [2].

3. Media feature combinations and capabilities

3.1 Media features

This memo assumes that individual media feature values are simple atomic values:

- Boolean values.

- Enumerated values.

- Text string values (treated as atomic entities, like enumerated value tokens).

- Numeric values (Integer or rational).

These values all have the property that they can be compared for equality ('='), and that numeric and ordered enumeration values can be compared for less-than and greater-than relationship ('<=', '>='). These basic comparison operations are used as the primitive building blocks for more comprehensive capability expressions.

3.2 Media feature collections and sets

Any single media feature value can be thought of as just one component of a feature collection that describes some instance of a resource (e.g. a printed document, a displayed image, etc.). Such a feature collection consists of a number of media feature tags (each per [3]) and associated feature values.

A feature set is a set containing a number of feature collections. Thus, a feature set can describe a number of different data resource instances. These can correspond to different treatments of a single data resource (e.g. different resolutions used for printing a given document), a number of different data resources subjected to a common treatment (e.g. the range of different images that can be rendered on a given display), or some combination of these (see examples below).

Thus, a description of a feature set can describe the capabilities of a data resource or some entity that processes or renders a data resource.

3.3 Media feature set descriptions

A feature set may be unbounded. For example, in principle, there is no limit on the number of different documents that may be output using a given printer. But for practical use, a feature set description must be finite.

The general approach to describing feature sets is to start from the assumption that anything is possible; i.e. the feature set contains all possible document instances (feature collections). Then constraints are applied that progressively remove document instances from this set; e.g. for a monochrome printer, all document instances that use colour are removed, or for a document that must be rendered at some minimum resolution, all document instances with lesser resolutions are removed from the set. The mechanism used to remove document instances from the set is the mathematical idea of a "relation"; i.e. a Boolean function (a "predicate") that takes a feature collection parameter and returns a Boolean value that is TRUE if the feature collection describes an acceptable document instance, or FALSE if it describes one that is excluded.

```
                    P(C)
       P(C) = TRUE <- : -> P(C) = FALSE
                      :
           +----------:----------+  This box represents some
           |          :          |  set of feature collections (C)
           | Included : Excluded |  that is constrained by the
           |          :          |  predicate P.
           +----------:----------+
                      :
```

The result of applying a series of such constraints is a smaller set of feature collections that represent some media handling capability. Where the individual constraints are represented by predicates that each describe some media handling capability, the combined effect of these constraints is some subset of the individual constraint capabilities that can be represented by a predicate that is the logical-AND of the individual constraint predicates.

3.4 Media feature combination scenario

3.4.1 Data resource options

The following expression uses the syntax introduced later to describe a data resource that can be displayed either:

(a) as a 750x500 pixel image using 15 colours, or

(b) at 150dpi on an A4 page.

```
(| (& (pix-x=750) (pix-y=500) (color=15) )
   (& (dpi>=150) (papersize=iso-A4) ) )
```

3.4.2 Recipient capabilities

The following expression describes a receiving system that has:

(a) a screen capable of displaying 640*480 pixels and 16 million colours (24 bits per pixel), 800*600 pixels and 64 thousand colours (16 bits per pixel) or 1024*768 pixels and 256 colours (8 bits per pixel), or

(b) a printer capable of rendering 300dpi on A4 paper.

Note that this expression says noting about the colour or grey-scale capabilities of the printer. In the scheme presented here, it is presumed to be unconstrained in this respect (or, more realistically, any such constraints are handled out-of-band by anyone sending to this recipient).

```
(| (& (| (& (pix-x<=640)  (pix-y<=480) (color<=16777216) )
         (& (pix-x<=800)  (pix-y<=600) (color<=65535) )
         (& (pix-x<=1024) (pix-y<=768) (color<=256) ) )
      (media=screen) )
   (& (dpi=300)
      (media=stationery) (papersize=iso-A4) ) )
```

3.4.3 Combined options

The following example describes the range of document representations available when the resource described in the first example above is sent to the recipient described in the second example. This is the result of combining their capability feature sets:

```
(| (& (pix-x=750) (pix-y=500) (color=15) )
   (& (dpi=300) (media=stationery) (papersize=iso-A4) ) )
```

The feature set described by this expression is the intersection of the sets described by the previous two capability expressions.

3.5 Feature set predicates

There are many ways of representing a predicate. The ideas in this memo were inspired by the programming language Prolog [5], and its use of predicates to describe sets of objects.

For the purpose of media feature description in networked application protocols, the format used for LDAP search filters [7,8] has been adopted, because it is a good match for the requirements of capability identification, and has a very simple structure that is easy to parse and process.

Observe that a feature collection is similar to a directory entry, in that it consists of a collection of named values. Further, the semantics of the mechanism for selecting feature collections from a feature set is in many respects similar to selection of directory entries from a directory.

A feature set predicate used to describe media handling capabilities is implicitly applied to some feature collection. Within the predicate, members of the feature collection are identified by their feature tags, and are compared with known feature values. (Compare with the way an LDAP search filter is applied to a directory entry, whose members are identified by attribute type names, and compared with known attribute values.)

Differences between directory selection (per [7]) and feature set selection are:

. Directory selection provides substring-, approximate- and extensible- matching for attribute values. Directory selection may also be based on the presence of an attribute without regard to its value.

. Directory selection provides for matching rules that test for the presence or absence of a named attribute type.

. Directory selection provides for matching rules which are dependent upon the declared data type of an attribute value.

. Feature selection provides for the association of a quality value with a feature predicate as a way of ranking the selected value collections.

The idea of substring matching does not seem to be relevant to feature set selection, and is excluded from these proposals.

Testing for the presence of a feature may be useful in some circumstances, but does not sit comfortably within the semantic framework. Feature sets are described by implied universal quantification over predicates, and the absence of reference to a given feature means the set is not constrained by that feature. Against this, it is difficult to define what might be meant by "presence" of a feature, so the "test for presence" option is not included in these proposals. An effect similar to testing for the presence of a feature can be achieved by a Boolean-valued feature.

The idea of extensible matching and matching rules dependent upon data types are facets of a problem not addressed by this memo, but which do not necessarily affect the feature selection syntax. An aspect which might have a bearing on the syntax would be a requirement to specify a matching rule explicitly as part of a selection expression.

4. Feature set representation

The foregoing sections have described a framework for defining feature sets with predicates applied to feature collections. This section presents a concrete representation for feature set predicates.

4.1 Textual representation of predicates

The text representation of a feature set is based on RFC 2254 "The String Representation of LDAP Search Filters" [8], excluding those elements not relevant to feature set selection (discussed above), and adding elements specific to feature set selection (e.g. options to associate quality values with predicates).

The format of a feature predicate is defined by the production for "filter" in the following, using the syntax notation and core rules of [10]:

```
filter     =  "(" filtercomp *( ";" parameter ) )"
parameter  =  "q" "=" qvalue
           /  ext-param "=" ext-value
qvalue     =  ( "0" [ "." 0*3DIGIT ] )
           /  ( "1" [ "." 0*3("0") ] )
ext-param  =  ALPHA *( ALPHA / DIGIT / "-" )
ext-value  =  <parameter value, according to the named parameter>
filtercomp =  and / or / not / item
and        =  "&" filterlist
or         =  "|" filterlist
not        =  "!" filter
filterlist =  1*filter
item       =  simple / set / ext-pred
set        =  attr "=" "[" setentry *( "," setentry ) "]"
setentry   =  value "/" range
range      =  value ".." value
simple     =  attr filtertype value
filtertype =  equal / greater / less
equal      =  "="
greater    =  ">="
less       =  "<="
attr       =  ftag
value      =  fvalue
ftag       =  <Feature tag, as defined in [3]>
fvalue     =  number / token / string
number     =  integer / rational
integer    =  1*DIGIT
rational   =  1*DIGIT "." 1*DIGIT
token      =  ALPHA *( ALPHA / DIGIT / "-" )
string     =  DQUOTE *(%x20-21 / %x23-7E) DQUOTE
              ; quoted string of SP and VCHAR without DQUOTE
ext-pred   =  <Extension constraint predicate, not defined here>
```

(Subject to constraints imposed by the protocol that carries a feature predicate, whitespace characters may appear between any pair of syntax elements or literals that appear on the right hand side of these productions.)

As described, the syntax permits parameters (including quality values) to be attached to any "filter" value in the predicate (not just top-level values). Only top-level quality values are recognized. If no explicit quality value is given, a value of '1.0' is applied.

NOTE

The flexible approach to quality values and other parameter values in this syntax has been adopted for two reasons: (a) to make it easy to combine separately constructed feature predicates, and (b) to provide an extensible tagging mechanism for possible future use (for example, to incorporate a conceivable requirement to explicitly specify a matching rule).

4.2 Named and auxiliary predicates

Named and auxiliary predicates can serve two purposes:

(a) making complex predicates easier to write and understand, and

(b) providing a possible basis for naming and registering feature sets.

[[[TODO: Decide how to treat named predicates. Support for named predicates in the capability syntax has not (currently) been made a requirement. However, its inclusion as an option may be useful for publication purposes, even if not used in actual protocol elements.]]]

4.2.1 Defining a named predicate

A named predicate definition has the following form:

```
named-pred =  "(" fname *pname ")" ":-" filter
fname      =  ftag        ; Feature predicate name
pname      =  token       ; Formal parameter name
```

'fname' is the name of the predicate.

'pname' is the name of a formal parameter which may appear in the predicate body, and which is replaced by some supplied value when the predicate is invoked.

'filter' is the predicate body. It may contain references to the formal parameters, and may also contain references to feature tags and other values defined in the environment in which the predicate is invoked. References to formal parameters may appear anywhere where a reference to a feature tag ('ftag') is permitted by the syntax for 'filter'.

The only specific mechanism defined by this memo for introducing a named predicate into a feature set definition is the "auxiliary predicate" described later. Specific negotiating protocols or other memos may define other mechanisms.

NOTE

There has been some suggestion of creating a registry for feature sets as well as individual feature values. Such a registry might be used to introduce named predicates corresponding to these feature sets into the environment of a capability assertion. Further discussion of this idea is beyond the scope of this memo.

4.2.2 Invoking named predicates

Assuming a named predicate has been introduced into the environment of some other predicate, it can be invoked by a filter 'ext-pred' of the form:

```
ext-pred   =  fname *param
param      =  expr
```

The number of parameters must match the definition of the named predicate that is invoked.

4.2.3 Auxiliary predicates in a filter

A auxiliary predicate is attached to a filter definition by the following extension to the "filter" syntax:

```
filter     =/ "(" filtercomp *( ";" parameter ) ")"
              "where" 1*( named-pred ) "end"
```

The named predicates introduced by "named-pred" are visible from the body of the "filtercomp" of the filter to which they are attached, but are not visible from each other. They all have access to the same environment as "filter", plus their own formal parameters. (Normal scoping rules apply: a formal parameter with the same name as a value in the environment of "filter" effectively hides the environment value from the body of the predicate to which it applies.)

NOTE

Recursive predicates are not permitted. The scoping rules should ensure this.

4.3 Feature set definition examples

The following sub-sections give examples of feature predicates that describes a number of image size and resolution combinations.

4.3.1 Single predicate

```
(| (& (Pix-x=1024)
      (Pix-y=768)
      (| (& (Res-x=150) (Res-y=150) )
         (& (Res-x=150) (Res-y=300) )
         (& (Res-x=300) (Res-y=300) )
         (& (Res-x=300) (Res-y=600) )
         (& (Res-x=600) (Res-y=600) ) )
   (& (Pix-x=800)
      (Pix-y=600)
      (| (& (Res-x=150) (Res-y=150) )
         (& (Res-x=150) (Res-y=300) )
         (& (Res-x=300) (Res-y=300) )
         (& (Res-x=300) (Res-y=600) )
         (& (Res-x=600) (Res-y=600) ) ) ;q=0.9
   (& (Pix-x=640)
      (Pix-y=480)
      (| (& (Res-x=150) (Res-y=150) )
         (& (Res-x=150) (Res-y=300) )
         (& (Res-x=300) (Res-y=300) )
         (& (Res-x=300) (Res-y=600) )
         (& (Res-x=600) (Res-y=600) ) ) ;q=0.8
```

4.3.2 Predicate with auxiliary predicate

```
(| (& (Pix-x=1024) (Pix-y=768) (Res Res-x Res-y) )
   (& (Pix-x=800)  (Pix-y=600) (Res Res-x Res-y) );q=0.9
   (& (Pix-x=640)  (Pix-y=480) (Res Res-x Res-y) );q=0.8 )
where
(Res Res-x Res-y) :-
   (| (& (Res-x=150) (Res-y=150) )
      (& (Res-x=150) (Res-y=300) )
      (& (Res-x=300) (Res-y=300) )
      (& (Res-x=300) (Res-y=600) )
      (& (Res-x=600) (Res-y=600) ) )
end
```

Note that the formal parameters of "Res", "Res-x" and "Res-y", prevent the body of the named predicate from referencing similarly-named feature values.

5. Processing feature set descriptions

This section addresses some issues that may arise when using feature set predicates as part of some content negotiation or file selection protocol.

5.1 Matching feature sets

Matching a feature set to some given feature collection is esentially very straightforward: the feature set predicate is simply evaluated for the given feature collection, and the result (TRUE or FALSE) indicates whether the feature collection matches the capabilities, and the associated quality value can be used for selecting among alternative feature collections.

Matching a feature set to some other feature set is less straightforward. Here, the problem is to determine whether or not there is at least one feature collection that matches both feature sets (e.g. is there an overlap between the feature capabilities of a given file format and the feature capabilities of a given recipient?)

This feature set matching is accomplished by logical manipulation of the predicate expressions as described in the following sections.

For this procedure to work reliably, the predicates must be reduced to a canonical form. One such form is "clausal form", and procedures for converting general expressions in predicate calculus are given in [5] (section 10.2), [11] (section 2.13), [12] (chapter 4) and [13] (section 5.3.2).

"Clausal form" for a predicate is similar to "conjunctive normal form" for a proposition, which consists of a conjunction (logical ANDs) of disjunctions (logical ORs). A related form that is better suited to feature set matching is "disjunctive normal form", which consists of a logical disjunction (OR) of conjunctions (ANDs). In this form, it is sufficient to show that at least one of the disjunctions can be satisfied by some feature collection.

A syntax for disjunctive normal form is:

```
filter     =  orlist
orlist     =  "(" "|" andlist ")" / term
andlist    =  "(" "&" termlist ")" / term
termlist   =  1*term
term       =  "(" "!" simple ")" / simple
```

where "simple" is as described previously in section 6.1. Thus, the canonicalized form has at most three levels: an outermost "(|...)" disjunction of "(&...)" conjunctions of possibly negated feature value tests.

NOTE (a theoretical diversion):

Is this consideration of "clausal form" really required? After all, the feature predicates are just Boolean expressions, aren't they?

Well, no. A feature predicate is a Boolean expression containing primitive feature value tests (comparisons), represented by 'item' in the feature predicate syntax. If these tests could all be assumed to be independently TRUE or FALSE, then each could be regarded as an atomic proposition, and the whole predicate could be dealt with according to the (relatively simple) rules of Propositional Calculus.

But, in general, the same feature tag may appear in more than one predicate 'item', so the tests cannot be regarded as independent. Indeed, interdependence is needed in any meaningful application of feature set matching, and it is important to capture these dependencies (e.g. does the set of resolutions that a sender can supply overlap the set of resolutions that a recipient can handle?). Thus, we have to deal with elements of the Predicate Calculus, with its additional rules for algebraic manipulation.

This section aims to show that these additional rules are more unfamiliar than complicated. In practice, the way that feature predicates are constructed and used actually avoids some of the complexity of dealing with fully-generalized Predicate Calculus.

5.1.1 Feature set matching strategy

The overall strategy for matching feature sets, expanded in the following sections, is:

1. Formulate the feature set match hypothesis.

2. Replace "set" expressions with equivalent comparisons.

3. Eliminate logical negations, and express all feature comparisons in terms of just four comparison operators

4. Reduce the hypothesis to canonical disjunctive normal form (a disjunction of conjunctions).

5. For each of the conjunctions, attempt to show that it can be satisfied by some feature collection. Any that cannot be satisfied are discarded.

   5.1 Separate the feature value tests into independent groups, such that each group contains tests involving just one feature value. That is: no group contains a predicate involving any feature tag that also appears in a predicate in some other group.

   5.2 For each group, merge the various constraints to a minimum form. This process either yields a reduced expression for the allowable range of feature values, or an indication that no value can satisfy the constraints (in which case the corresponding conjucntion can never be satisfied).

6. If the remaining disjunction is non-empty, then the constraints are shown to be satisfiable. Further, it can be used as a statement of the resulting feature set for possible further matching operations.

5.1.2 Formulating the goal predicate

A formal statement of the problem we need to solve can be given as: given two feature set predicates, '(P x)' and '(Q x)', where 'x' is some feature collection, we wish to establish the truth or otherwise of the proposition:

```
EXISTS(x) : (P x) AND (Q x)
```

i.e. does there exist a feature collection 'x' that satisfies both predicates, 'P' and 'Q'?

Then, if feature sets to be matched are described by predicates 'P' and 'Q', the problem is to determine if there is any feature set satisfying the goal predicate:

```
(& P Q)
```

i.e. to determine whether the set thus described is non-empty.

5.1.3 Replace set expressions

Replace all "set" instances in the goal predicate with equivalent "simple" forms:

```
T = [ E1, E2, ... En ]  -->  (| (T=[E1]) (T=[E2]) ... (T=[En]) )
(T=[R1..R2])            -->  (& (T>=R1) (T<=R2) )
(T=[E])                 -->  (T=E)
```

5.1.4 Replace comparisons and logical negations

The predicates are derived from the syntax described previously, and contain primitive value testing functions '=', '<=', '>='. The primitive tests have a number of well known properties that are exploited to reach a useful conclusion; e.g.

```
(A = B)  & (B = C)  => (A = C)
(A <= B) & (B <= C) => (A <= C)
```

These rules form a core body of logic statements against which the goal predicate can be evaluated. The form in which these statements are expressed is important to realizing an effective predicate matching algorithm (i.e. one that doesn't loop or fail to find a valid result). The first step in formulating these rules is to simplify the framework of primitive predicates.

The primitive predicates from which feature set definitions are constructed are '=', '<=' and '>='. Observe that, given any pair of feature values, the relationship between them must be exactly one of the following:

```
(LT a b): 'a' is less than 'b'.
(EQ a b): 'a' is equal to 'b'.
(GT a b): 'a' is greater than 'b'.
(NE a b): 'a' is not equal and not related to 'b'.
```

(The final case arises when two values are compared for which no ordering relationship is defined, and the values are not equal; e.g. two unequal string values.)

These four cases can be captured by a pair of primitive predicates:

```
(LE a b): 'a' is less than or equal to 'b'.
(GE a b): 'a' is greater than or equal to 'b'.
```

The four cases described above are prepresented by the following combinations of primitive predicate values:

(LE a b)	(GE a b)	relationship
TRUE	FALSE	(LT a b)
TRUE	TRUE	(EQ a b)
FALSE	TRUE	(GT a b)
FALSE	FALSE	(NE a b)

Thus, the original 3 primitive tests can be translated to combinations of just LE and GE, reducing the number of additional relationships that must be subsequently captured:

```
(a <= b)  -->  (LE a b)
(a >= b)  -->  (GE a b)
(a = b)   -->  (& (LE a b) (GE a b) )
```

Further, logical negations of the original 3 primitive tests can be eliminated by the introduction of 'not-greater' and 'not-less' primitives

```
(NG a b)  ==  (! (GE a b) )
(NL a b)  ==  (! (LE a b) )
```

using the following transformation rules:

```
(! (a = b) )   -->  (| (NL a b) (NG a b) )
(! (a <= b) )  -->  (NL a b)
(! (a >= b) )  -->  (NG a b)
```

Thus, we have rules to transform all comparisons and logical negations into combinations of just 4 relational operators.

5.1.5 Conversion to canonical form

Expand bracketed disjunctions, and flatten bracketed conjunctions and disjunctions:

```
(& (| A1 A2 ... Am ) B1 B2 ... Bn )
  -->  (| (& A1 B1 B2 ... Bn )
          (& A2 B1 B2 ... Bn )
           :
          (& Am B1 B2 ... Bn ) )
(& (& A1 A2 ... Am ) B1 B2 ... Bn )
  -->  (& A1 A2 ... Am B1 B2 ... Bn )
(| (| A1 A2 ... Am ) B1 B2 ... Bn )
  -->  (| A1 A2 ... Am B1 B2 ... Bn )
```

The result is a "disjunctive normal form", a disjunction of conjunctions:

```
(| (& S11 S12 ... )
   (& S21 S22 ... )
    :
   (& Sm1 Sm2 ... Smn ) )
```

where the "Sij" elements are simple feature comparison forms constructed during the step at section 7.1.4. Each term within the top-level "(|...)" construct represents a single possible feature set that satisfies the goal. Note that the order of entries within the top-level '(|...)', and within each '(&...)', is immaterial.

From here on, each conjunction '(&...)' is processed separately. Only one of these needs to be satisfiable for the original goal to be satisfiable.

(A textbook conversion to clausal form [5,11] uses slightly different rules to yield a "conjunctive normal form".)

5.1.6 Grouping of feature predicates

> NOTE: remember that from here on, each conjunction is treated separately.

Each simple feature predicate contains a "left-hand" feature tag and a "right-hand" feature value with which it is compared.

To arrange these into independent groups, simple predicates are grouped according to their left hand feature tag ('f').

5.1.7 Merge single-feature constraints

Within each group, apply the predicate simplification rules given below to eliminate redundant single-feature constraints. All single-feature predicates are reduced to an equality or range constraint on that feature, possibly combined with a number of non-equality statements.

If the constraints on any feature are found to be contradictory (i.e. resolved to FALSE according to the applied rules), the current conjunction is removed from the feature set description. Otherwise, the resulting description is a minimal form of the particular conjunction of the feature set definition.

5.1.7.1 Rules for simplifying ordered values

These rules are applicable where there is an ordering relationship between the given values 'a' and 'b':

```
(LE f a)  (LE f b)      -->  (LE f a),   a<=b
                             (LE f b),   otherwise
(LE f a)  (GE f b)      -->  FALSE,      a<b
(LE f a)  (NL f b)      -->  FALSE,      a<=b
(LE f a)  (NG f b)      -->  (LE f a),   a<b
                             (NG f b),   otherwise

(GE f a)  (GE f b)      -->  (GE f a),   a>=b
                             (GE f b),   otherwise
(GE f a)  (NL f b)      -->  (GE f a)    a>b
                             (NL f b),   otherwise
(GE f a)  (NG f b)      -->  FALSE,      a>=b

(NL f a)  (NL f b)      -->  (NL f a),   a>=b
                             (NL f b),   otherwise
(NL f a)  (NG f b)      -->  FALSE,      a>=b

(NG f a)  (NG f b)      -->  (NG f a),   a<=b
                             (NG f b),   otherwise
```

5.1.7.2 Rules for simplifying unordered values

These rules are applicable where there is no ordering relationship applicable to the given values 'a' and 'b':

```
   (LE f a)  (LE f b)      -->  (LE f a),   a=b
                                FALSE,      otherwise
   (LE f a)  (GE f b)      -->  FALSE,      a!=b
   (LE f a)  (NL f b)      -->  (LE f a)    a!=b
                                FALSE,      otherwise
   (LE f a)  (NG f b)      -->  (LE f a),   a!=b
                                FALSE,      otherwise

   (GE f a)  (GE f b)      -->  (GE f a),   a=b
                                FALSE,      otherwise
   (GE f a)  (NL f b)      -->  (GE f a)    a!=b
                                FALSE,      otherwise
   (GE f a)  (NG f b)      -->  (GE f a)    a!=b
                                FALSE,      otherwise

   (NL f a)  (NL f b)      -->  (NL f a),   a=b
   (NL f a)  (NG f b)      -->  (NL f a),   a=b

   (NG f a)  (NG f b)      -->  (NG f a),   a=b
```

[[[TODO: model the above system to confirm that it is complete and does indeed work properly in all cases.]]]

5.2 Effect of named predicates

The preceding procedures can be extended to deal with named predicates simply by instantiating (i.e. substituting) the predicates wherever they are invoked, before performing the conversion to disjunctive normal form. In the absence of recursive predicates, this procedure is guaranteed to terminate.

When substituting the body of a precdicate at its point of invocation, instances of formal parameters within the predicate body must be replaced by the corresponding actual parameter from the point of invocation.

5.3 Unit designations

In some exceptional cases, there may be differing conventions for the units of measurement of a given feature. For example, resolution is commonly expressed as dots per inch (dpi) or dots per centimetre (dpcm) in different applications (e.g. printing vs faxing).

In such cases, a unit designator may be appended to a feature value according to the conventions indicated below (see also [3]). These considerations apply only to features with numeric values.

Every feature tag has a standard unit of measurement. Any expression of a feature value that uses this unit is given without a unit designation -- this is the normal case. When the feature value is expressed in some other unit, a unit designator is appended to the numeric feature value.

The registration of a feature tag indicates the standard unit of measurement for a feature, and also any alternate units and corresponding unit designators that may be used, according to [3].

Thus, if the standard unit of measure for resolution is 'dpcm', then the feature predicate '(res=200)' would be used to indicate a resolution of 200 dots-per-centimetre, and '(res=72dpi)' might be used to indicate 72 dots-per-inch.

Unit designators are accommodated by the following extension to the feature predicate syntax:

```
fvalue     /= number *WSP token
```

When performing feature set matching, feature comparisons with and without unit designators, or feature comparisons with different unit designators, are treated as if they were different features. Thus, the feature predicate '(res=200)' would not, in general, fail to match with the predicate '(res=200dpi)'.

NOTE:

A protocol processor with specific knowledge of the feature and units concerned might recognize the relationship between the feature predicates in the above example, and fail to match these predicates.

This appears to be a natural behaviour in this simple example, but can cause additional complexity in more general cases. Accordingly, this is not considered to be required or normal behaviour. It is presumed that in general, the application concerned will ensure consistent feature processing by adopting a consistent unit for any given feature.

5.4 Unknown feature value data types

[[Discuss issues of specific features which may have feature-specific comparison rules, as opposed to generic Booleans, enumerations, strings and numbers which use comparison rules independent of the feature concerned.]]

[[[TODO]]]

5.5 Worked example

[[[TODO]]]

5.6 Algorithm source code

[[[TODO]]]

6. Security considerations

Some security considerations for content negotiation are raised in [1,2,3].

The following are primary security concerns for capability identification mechanisms:

- Unintentional disclosure of private information through the announcement of capabilities or user preferences.

- Disruption to system operation caused by accidental or malicious provision of incorrect capability information.

- Use of a capability identification mechanism might be used to probe a network (e.g. by identifying specific hosts used, and exploiting their known weaknesses).

The most contentious security concerns are raised by mechanisms which automatically send capability identification data in response to a query from some unknown system. Use of directory services (based on LDAP [7], etc.) seem to be less problematic because proper authentication mechanisms are available.

Mechanisms which provide capability information when sending a message are less contentious, presumably because some intention can be inferred that person whose details are disclosed wishes to communicate with the recipient of those details. This does not, however, solve problems of spoofed supply of incorrect capability information.

The use of format converting gateways may prove problematic because such systems would tend to defeat any message integrity and authenticity checking mechanisms that are employed.

7. Copyright

8. Acknowledgements

My thanks to Larry Masinter for demonstrating to me the breadth of the media feature issue, and encouraging me to air my early ideas.

Early discussions of ideas on the IETF-HTTP and IETF-FAX discussion lists led to useful inputs also from Koen Holtman, Ted Hardie and Dan Wing.

The debate later moved to the IETF conneg WG mailing list, where Al Gilman was particularly helpful in helping me to refine the feature set algebra. Ideas for dealing with preferences and specific units were suggested by Larry Masinter.

This work was supported by Content Technologies Ltd and 5th Generation Messaging Ltd.

9. References

[1] "Scenarios for the Delivery of Negotiated Content"
T. Hardie, NASA Network Information Center
Internet draft: <draft-ietf-http-negotiate-scenario-02.txt>
Work in progress, November 1997.

[2] "Requirements for protocol-independent content negotiation"
G. Klyne, Integralis Ltd.
Internet draft: <draft-ietf-conneg-requirements-00.txt>
Work in progress, March 1998.

[3] "Media Feature Tag Registration Procedure"
Koen Holtman, TUE
Andrew Mutz, Hewlett-Packard
Ted Hardie, NASA
Internet draft: <draft-ietf-conneg-feature-reg-03.txt>
Work in progress, July 1998.

[4] "Notes on data structuring"
C. A. R. Hoare,
in "Structured Programming"
Academic Press, APIC Studies in Data Processing No. 8
ISBN 0-12-200550-3 / 0-12-200556-2
1972.

[5] "Programming in Prolog" (2nd edition)
W. F. Clocksin and C. S. Mellish,
Springer Verlag
ISBN 3-540-15011-0 / 0-387-15011-0

1984.

[6] "Media Features for Display, Print, and Fax"
     Larry Masinter, Xerox PARC
     Koen Holtman, TUE
     Andrew Mutz, Hewlett-Packard
     Dan Wing, Cisco Systems
     Internet draft: <draft-masinter-media-features-02.txt>
     Work in progress, January 1998.

[7] RFC 2251, "Lightweight Directory Access Protocol (v3)"
     M. Wahl, Critical Angle Inc.
     T. Howes, Netscape Communications Corp.
     S. Kille, Isode Limited
     December 1997.

[8] RFC 2254, "The String Representation of LDAP Search Filters"
     T. Howes, Netscape Communications Corp.
     December 1997.

[9] RFC 2068, "Hyptertext Transfer Protocol -- HTTP/1.1"
     R. Fielding, UC Irvine
     J. Gettys,
     J. Mogul, DEC
     H. Frytyk,
     T. Berners-Lee, MIT/LCS
     January 1997.

[10] RFC 2234, "Augmented BNF for Syntax Specifications: ABNF"
     D. Crocker (editor), Internet Mail Consortium
     P. Overell, Demon Internet Ltd.
     November 1997.

[11] "Logic, Algebra and Databases"
     Peter Gray
     Ellis Horwood Series: Computers and their Applications
     ISBN 0-85312-709-3/0-85312-803-3 (Ellis Horwood Ltd)
     ISBN 0-470-20103-7/0-470-20259-9 (Halstead Press)
     1984.

[12] "Elementary Logics: A procedural Perspective
     Dov Gabbay
     Prentice Hall, Series in computer science
     ISBN 0-13-726365-1
     1998.

[13] "Logic and its Applications"
     Edmund Burk and Eric Foxley
     Prentice Hall, Series in computer science

ISBN 0-13-030263-5
1996.

10. Author's address

Graham Klyne

Content Technologies Ltd.	5th Generation Messaging Ltd.
Forum 1	5 Watlington Street
Station Road	Nettlebed
Theale	Henley-on-Thames
Reading, RG7 4RA	RG9 5AB
United Kingdom	United Kingdom.
Telephone: +44 118 930 1300	+44 1491 641 641
Facsimile: +44 118 930 1301	+44 1491 641 611

E-mail: GK@ACM.ORG