Network Working Group
Request for Comments: 2533
Category: Standards Track

G. Klyne
Content Technologies/5GM

March 1999

A Syntax for Describing Media Feature Sets

Status of this Memo

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 which 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 feature registration 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.

An algorithm for feature set matching is also described here.

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.

An algorithm for feature set matching is also described here.

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 describes an algorithm for feature set matching.

Section 6 discusses some additional media feature description and processing issues that may be viewed as extensions to the core framework.

Section 7 contains a worked example of feature set matching, and some additional explanatory comments spurred by issues arising from applying this framework to fascimile transmissions.

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 RFC 2119.

> 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/

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.

NOTE: this term is used slightly differently by earlier work on Transparent Content Negotiation in HTTP [4].

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 memo 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.

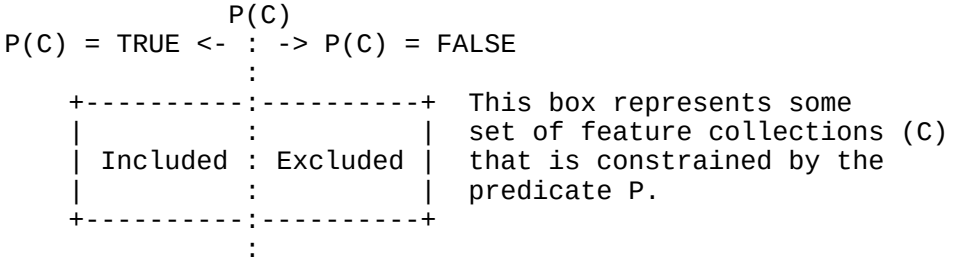
A feature set is a set containing a number of feature collections. Thus, a feature set can describe a number of different data resource instances. These can correspond to different treatments of a single data resource (e.g. different resolutions used for printing a given document), a number of different 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 to be practically useful, 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.



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

This section develops some example scenarios, introducing the notation that is defined formally in section 4.

3.4.1 Data resource options

The following expression describes 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.

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

Note that this expression says nothing 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).

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) (ua-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 descriptions 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.

3.5.1 Comparison with directory search filters

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.)

For example, in:

```
(& (dpi>=150) (papersize=iso-A4) )
```

the tokens 'dpi' and 'papersize' are feature tags, and '150' and 'iso-A4' are feature values. (In a corresponding LDAP search filter, they would be directory entry attribute types and attribute values.)

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

- Directory selection provides substring-, approximate- and extensible- matching for attribute values. Such matching is not provided for feature set selection.
- Directory selection may be based on the presence of an attribute without regard to its value. Within the semantic framework described by this document, Boolean-valued feature tests can be used to provide a similar effect.

- 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.

Within the semantic framework described by this document, Boolean-valued feature tests can be used where presence tests would be used in a directory search filter.

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 that might bear on the syntax would be specification of an explicit matching rule as part of a selection expression.

3.6 Describing preferences

A convenient way to describe preferences is by numeric "quality values".

It has been suggested that numeric quality values are potentially misleading if used as more than just a way of ranking options. For the purposes of this memo, ranking of options is sufficient.

Numeric quality values in the range 0 to 1, with up to 3 fractional digits, are used to rank feature sets according to preference. Higher values are preferred over lower values, and equal values are presumed to be equally preferred. Beyond this, the actual number used has no significance defined here. Arithmetic operations on quality values are likely to produce unpredictable results unless appropriate semantics have been defined for the context where such operations are used.

In the absence of any explicitly applied quality value, a value of "1" is assumed.

Using the notation defined later, a quality value may be attached to any feature set predicate sub-expression:

```
(| (& (pix-x=750) (pix-y=500) (color=15) );q=0.8
   (& (dpi>=150) (papersize=iso-A4) )     ;q=0.7 )
```

Section 3.7 below explains that quality values attached to sub-expressions are not always useful.

NOTE: the syntax for quality values used here taken from that defined for HTTP 'Accept:' headers in RFC 2068 [9], section 3.9. However, the use of quality values defined here does not go as far as that defined in RFC 2068.

3.7 Combining preferences

The general problem of describing and combining preferences among feature sets is very much more complex than simply describing allowable feature sets. For example, given two feature sets:

```
(& (a1);q=0.8 (b1);q=0.7 )
(& (a2);q=0.5 (b2);q=0.9 )
```

where:
feature a1 is preferred over a2
feature b2 is preferred over b1

Which of these feature sets is preferred? In the absence of additional information or assumptions, there is no generally satisfactory answer to this.

The proposed resolution of this issue is simply to say that no rules are provided for combining preference information. Applied to the above example, any preference information about (a1) in relation to (a2), or (b1) in relation to (b2) is not presumed to convey information about preference of (& (a1) (b1)) in relation to (& (a2) (b2)).

In practical terms, this restricts the application of preference information to top-level predicate clauses. A top-level clause completely defines an allowable feature set; clauses combined by logical-AND operators cannot be top-level clauses (see canonical format for feature set predicates, described later).

NOTE: This memo does not apply specific meaning to quality values or rules for combining them. Application of such meanings and rules is not prohibited, but is seen as an area for continuing research and experimentation.

An example of a design that uses extended quality value semantics and combining operations is "Transparent Content Negotiation in HTTP" [4]. Other work that also extends quality values is the content negotiation algorithm in the Apache HTTP server [14].

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 RFC 2234 [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 RFC 2506 [3]>
fvalue     =  Boolean / number / token / string
Boolean    =  "TRUE" / "FALSE"
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 Interpretation of feature predicate syntax

A feature set predicate is described by the syntax production for 'filter'.

4.2.1 Filter syntax

A 'filter' is defined as either a simple feature comparison ('item', see below) or a composite filter ('and', 'or', 'not'), decorated with optional parameter values (including "q=qvalue").

A composite filter is a logical combination of one or more 'filter' values:

(& f1 f2 ... fn)	is the logical-AND of the filter values 'f1', 'f2' up to 'fn'. That is, it is satisfied by any feature collection that satisfies all of the predicates represented by those filters.
(| f1 f2 ... fn)	is the logical-OR of the filter values 'f1', 'f2' up to 'fn'. That is, it is satisfied by any feature collection that satisfies at least one of the predicates represented by those filters.

(! f1)	is the logical negation of the filter value 'f1'. That is, it is satusfied by any feature collection that does NOT satisfy the predicate represented by 'f1'.

4.2.2 Feature comparison

A feature comparison is defined by the 'simple' option of the syntax production for 'item'. There are three basic forms:

(ftag=value)	compares the feature named 'ftag' (in some feature collection that is being tested) with the supplied 'value', and matches if they are equal. This can be used with any type of feaure value (numeric, Boolean, token or string).
(ftag<=value)	compares the numeric feature named 'ftag' with the supplied 'value', and matches if the feature is less than or equal to 'value'.
(ftag>=value)	compares the numeric feature named 'ftag' with the supplied 'value', and matches if the feature is greater than or equal to 'value'.

Less-than and greater-than tests may be performed with feature values that are not numeric but, in general, they amount to equality tests as there is no ordering relation on non-numeric values defined by this specification. Specific applications may define such ordering relations on specific feature tags, but such definitions are beyond the scope of (and not required for conformance to) this specification.

4.2.3 Feature tags

Feature tags conform to the syntax given in "Media Feature Tag Registration Procedure" [3]. Feature tags used to describe capabilities should be registered using the procedures described in that memo. Unregistered feature tags should be allocated in the "URI tree", as discussed in the media feature registration procedures memo [3].

If an unrecognized feature tag is encountered in the course of feature set predicate processing, it should be still be processed as a legitimate feature tag. The feature set matching rules are designed to allow new feature tags to be introduced without affecting the validity of existing capability assertions.

4.2.4 Feature values

A feature may have a number, Boolean, token or string value.

4.2.4.1 Boolean values

A Boolean is simply a token with two predefined values: "TRUE" and "FALSE". (Upper- or lower- case letters may be used in any combination.)

4.2.4.2 Numeric values

A numeric value is either a decimal integer, optionally preceded by a "+" or "-" sign, or rational number.

A rational number is expressed as "n/m", optionally preceded by a "+" or "-" sign. The "n" and "m" are unsigned decimal integers, and the value represented by "n/m" is "n" divided by "m". Thus, the following are all valid representations of the number 1.5:

```
3/2
+15/10
600/400
```

Thus, several rational number forms may express the same value. A canonical form of rational number is obtained by finding the highest common factor of "n" and "m", and dividing both "n" and "m" by that value.

A simple integer value may be used anywhere in place of a rational number. Thus, we have:

```
+5 is equivalent to +5/1 or +50/10, etc.
-2 is equivalent to -2/1 or -4/2, etc.
```

Any sign in a rational number must precede the entire number, so the following are not valid rational numbers:

```
3/+2, 15/-10      (**NOT VALID**)
```

4.2.4.3 Token values

A token value is any sequence of letters, digits and ’-’ characters that conforms to the syntax for ’token’ given above. It is a name that stands for some (unspecified) value.

4.2.4.4 String values

A string value is any sequence of characters enclosed in double quotes that conform to the syntax for 'string' given above.

The semantics of string defined by this memo are the same as those for a token value. But a string allows a far greater variety of internal formats, and specific applications may choose to interpret the content in ways that go beyond those given here. Where such interpretation is possible, the allowed string formats and the corresponding interpretations should be indicated in the media feature registration (per RFC 2506 [3]).

4.2.5 Notational conveniences

The 'set' option of the syntax production for 'item' is simply a shorthand notation for some common situations that can be expressed using 'simple' constructs. Occurrences of 'set' items can eliminated by applying the following identities:

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

Examples:

The expression:

```
( paper-size=[A4,B4] )
```

can be used to express a capability to print documents on either A4 or B4 sized paper.

The expression:

```
( width=[4..17/2] )
```

might be used to express a capability to print documents that are anywhere between 4 and 8.5 inches wide.

The set construct is designed so that enumerated values and ranges can be combined in a single expression, e.g.:

```
( width=[3,4,6..17/2] )
```

4.3 Feature set definition example

The following is an example of a feature predicate that describes a number of image size and resolution combinations, presuming the registration and use of 'Pix-x', 'Pix-y', 'Res-x' and 'Res-y' feature tags:

```
(| (& (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 )
```

5. Matching feature sets

This section presents a procedure for combining feature sets to determine the common feature collections to which they refer, if there are any. Making a selection from the possible feature collections (based on q-values or otherwise) is not covered here.

Matching a feature set to some given feature collection is essentially 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 sub-sections.

For this procedure to work reliably, the predicates must be reduced to a canonical form. The canonical form used here is "disjunctive normal form". 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 4.1. Thus, the canonicalized form has at most three levels: an outermost "(|...)" disjunction of "(&...)" conjunctions of possibly negated feature value tests.

NOTE: The usual canonical form for predicate expressions is "clausal form". Procedures for converting general predicate expressions are given in [5] (section 10.2), [11] (section 2.13) and [12] (section 5.3.2).

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

Is this consideration of canonical forms 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 some additional rules for algebraic manipulation.

A description of both the Propositional and Predicate calculi can be found in [12].

We aim to show that these additional rules are more unfamiliar than complicated. The construction and use of feature predicates actually avoids some of the complexity of dealing with fully-generalized Predicate Calculus.

5.1 Feature set matching strategy

The overall strategy for matching feature sets, expanded below, is:

1. Formulate the feature set match hypothesis.

2. Replace "set" expressions with equivalent comparisons.

3. Move logical negations "inwards", so that they are all applied directly to feature comparisons.

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

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

6. For each of the conjunctions, attempt to show that it can be satisfied by some feature collection.

   6.1 Separate the feature value tests into independent feature groups, such that each group contains tests involving just one feature tag. Thus, no predicate in a feature group contains a feature tag that also appears in some other group.

   6.2 For each feature 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 expression containing the value FALSE, which is an indication that no combination of feature values can satisfy the constraints (in which case the corresponding conjunction can never be satisfied).

7. If the remaining disjunction contains at least one satisfiable conjunction, then the constraints are shown to be satisfiable.

The final expression obtained by this procedure, if it is non-empty, can be used as a statement of the resulting feature set for possible further matching operations. That is, it can be used as a starting point for combining with additional feature set constraint predicate to determine a feature set that is constrained by the capabilities of several entities in a message transfer path.

NOTE: as presented, the feature matching process evaluates (and stores) all conjunctions of the disjunctive normal form before combining feature tag comparisons and eliminating unsatisfiable conjunctions. For low-memory systems an alternative approach is possible, in which each normal form conjunction is enumerated and evaluated in turn, with only those that are satisfiable being retained for further use.

5.2 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.

5.3 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)
```

5.4 Move logical negations inwards

The goal of this step is to move all logical negations so that they are applied directly to feature comparisons. During the following step, these logical negations are replaced by alternative comparison operators.

This is achieved by repeated application of the following transformation rules:

```
(! (& A1 A2 ... Am ) )  -->  (| (! A1 ) (! A2 ) ... (! Am ) )
(! (| A1 A2 ... Am ) )  -->  (& (! A1 ) (! A2 ) ... (! Am ) )
(! (! A ) )             -->  A
```

The first two rules are extended forms of De Morgan's law, and the third is elimination of double negatives.

5.5 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 to 'b', and is not less than
          or greater than '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.

5.6 Conversion to canonical form

NOTE: Logical negations have been eliminated in the previous step.

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 in "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 5.5. 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".)

5.7 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').

5.8 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 containing conjunction is not satisfiable and may be discarded. Otherwise, the resulting description is a minimal form of that particular conjunction of the feature set definition.

5.8.1 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
```

5.8.2 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
```

6. Other features and issues

6.1 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.

6.1.1 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 specifications 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.

6.1.2 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.

6.1.3 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.

6.1.4 Feature matching with 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.

6.1.5 Example

This example restates that given in section 4.3 using an auxiliary predicate named 'Res':

```
(| (& (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.

6.2 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 RFC 2506 [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 an application concerned will ensure consistent feature processing by adopting a consistent unit for any given feature.

6.3 Unknown feature value data types

This memo has dealt with feature values that have well-understood comparison properties: numbers, with equality, less-than, greater-than relationships, and other values with equality relationships only.

Some feature values may have comparison operations that are not covered by this framework. For example, strings containing multi-part version numbers: "x.y.z". Such feature comparisons are not covered by this memo.

Specific applications may recognize and process feature tags that are associated with such values. Future work may define ways to introduce new feature value data types in a way that allows them to be used by applications that do not contain built-in knowledge of their properties.

7. Examples and additional comments

7.1 Worked example

This example considers sending a document to a high-end black-and-white fax system with the following receiver capabilities:

```
   (& (dpi=[200,300])
      (grey=2) (color=0)
      (image-coding=[MH,MR]) )
```

Turning to the document itself, assume it is available to the sender in three possible formats, A4 high resolution, B4 low resolution and A4 high resolution colour, described by:

```
   (& (dpi=300)
      (grey=2)
      (image-coding=MR) )

   (& (dpi=200)
      (grey=2)
      (image-coding=[MH,MMR]) )

   (& (dpi=300) (dpi-xyratio=1)
      (color<=256)
      (image-coding=JPEG) )
```

These three image formats can be combined into a composite capability statement by a logical-OR operation (to describe format-1 OR format-2 OR format-3):

```
   (| (& (dpi=300)
         (grey=2)
         (image-coding=MR) )
      (& (dpi=200)
         (grey=2)
         (image-coding=[MH,MMR]) )
      (& (dpi=300)
         (color<=256)
         (image-coding=JPEG) ) )
```

The composite document description can be matched with the receiver capability description by combining the capability descriptions with a logical AND operation:

```
   (& (& (dpi=[200,300])
           (grey=2) (color=0)
         (image-coding=[MH,MR]) )
      (| (& (dpi=300)
            (grey=2)
            (image-coding=MR) )
         (& (dpi=200)
            (grey=2)
            (image-coding=[MH,MMR]) )
         (& (dpi=300)
```

```
               (color<=256)
               (image-coding=JPEG) ) ) )

   -->  Expand value-set notation:

      (& (& (| (dpi=200) (dpi=300) )
            (grey=2) (color=0)
            (| (image-coding=MH) (image-coding=MR) ) )
         (| (& (dpi=300)
               (grey=2)
               (image-coding=MR) )
            (& (dpi=200)
               (grey=2)
               (| (image-coding=MH) (image-coding=MMR) ) )
            (& (dpi=300)
               (color<=256)
               (image-coding=JPEG) ) ) )

   -->  Flatten nested '(&...)':

      (& (| (dpi=200) (dpi=300) )
         (grey=2) (color=0)
         (| (image-coding=MH) (image-coding=MR) )
         (| (& (dpi=300)
               (grey=2)
               (image-coding=MR) )
            (& (dpi=200)
               (grey=2)
               (| (image-coding=MH) (image-coding=MMR) ) )
            (& (dpi=300)
               (color<=256)
               (image-coding=JPEG) ) ) )

   -->  (distribute '(&...)' over inner '(|...)'):

      (& (| (dpi=200) (dpi=300) )
         (grey=2) (color=0)
         (| (image-coding=MH) (image-coding=MR) )
         (| (& (dpi=300) (grey=2) (image-coding=MR) )
            (& (dpi=200) (grey=2) (image-coding=MH) )
            (& (dpi=200) (grey=2) (image-coding=MMR) )
            (& (dpi=300) (color<=256) (image-coding=JPEG) ) ) )

   -->  continue to distribute '(&...)' over '(|...)', and flattening
        nested '(&...)' and '(|...)' ...:

      (| (& (dpi=200) (grey=2) (color=0) (image-coding=MH)
            (| (& (dpi=300) (grey=2) (image-coding=MR) )
```

```
               (& (dpi=200) (grey=2) (image-coding=MH) )
               (& (dpi=200) (grey=2) (image-coding=MMR) )
               (& (dpi=300) (color<=256) (image-coding=JPEG) ) ) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MR)
            (| (& (dpi=300) (grey=2) (image-coding=MR) )
               (& (dpi=200) (grey=2) (image-coding=MH) )
               (& (dpi=200) (grey=2) (image-coding=MMR) )
               (& (dpi=300) (color<=256) (image-coding=JPEG) ) ) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MH)
            (| (& (dpi=300) (grey=2) (image-coding=MR) )
               (& (dpi=200) (grey=2) (image-coding=MH) )
               (& (dpi=200) (grey=2) (image-coding=MMR) )
               (& (dpi=300) (color<=256) (image-coding=JPEG) ) ) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MR)
            (| (& (dpi=300) (grey=2) (image-coding=MR) )
               (& (dpi=200) (grey=2) (image-coding=MH) )
               (& (dpi=200) (grey=2) (image-coding=MMR) )
               (& (dpi=300) (color<=256) (image-coding=JPEG) ) ) ) )

   -->  ... until normal form is achieved:

      (| (& (dpi=200) (grey=2) (color=0) (image-coding=MH)
            (dpi=300) (grey=2) (image-coding=MR) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MR)
            (dpi=300) (grey=2) (image-coding=MR) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MH)
            (dpi=300) (grey=2) (image-coding=MR) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MR)
            (dpi=300) (grey=2) (image-coding=MR) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MH)
            (dpi=200) (grey=2) (image-coding=MH) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MR)
            (dpi=200) (grey=2) (image-coding=MH) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MH)
            (dpi=200) (grey=2) (image-coding=MH) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MR)
            (dpi=200) (grey=2) (image-coding=MH) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MH)
            (dpi=200) (grey=2) (image-coding=MMR) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MR)
            (dpi=200) (grey=2) (image-coding=MMR) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MH)
            (dpi=200) (grey=2) (image-coding=MMR) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MR)
            (dpi=200) (grey=2) (image-coding=MMR) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MH)
            (dpi=300) (color<=256) (image-coding=JPEG) ) ) )
         (& (dpi=200) (grey=2) (color=0) (image-coding=MR)
```

```
            (dpi=300) (color<=256) (image-coding=JPEG) ) ) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MH)
            (dpi=300) (color<=256) (image-coding=JPEG) ) ) )
         (& (dpi=300) (grey=2) (color=0) (image-coding=MR)
            (dpi=300) (color<=256) (image-coding=JPEG) ) )
```

--> Group terms in each conjunction by feature tag:

```
   (| (& (dpi=200) (dpi=300) (grey=2) (grey=2) (color=0)
         (image-coding=MH) (image-coding=MR) )
      (& (dpi=200) (dpi=300) (grey=2) (grey=2) (color=0)
         (image-coding=MR) (image-coding=MR) )
          :
         (etc.)
          :
      (& (dpi=300) (dpi=300) (grey=2) (color=0) (color<=256)
         (image-coding=MR) (image-coding=JPEG) ) )
```

--> Combine feature tag comparisons and eliminate unsatisfiable conjunctions:

```
   (| (& (dpi=300) (grey=2) (color=0) (image-coding=MR) )
      (& (dpi=200) (grey=2) (color=0) (image-coding=MH) ) )
```

Thus, we see that this combination of sender and receiver options can transfer a bi-level image, either at 300dpi using MR coding, or at 200dpi using MH coding.

Points to note about the feature matching process:

- The colour document option is eliminated because the receiver cannot handle either colour (indicated by '(color=0)') or JPEG coding.

- The high resolution version of the document with '(dpi=300)' must be sent using '(image-coding=MR)' because this is the only available coding of the image data that the receiver can use for high resolution documents. (The available 300dpi document codings here are MMR and MH, and the receiver capabilities are MH and MR.)

7.2 A note on feature tag scoping

This section contains some additional commentary on the interpretation of feture set predicates. It does not extend or modify what has been described previously. Rather, it attempts to clarify an area of possible misunderstanding.

The essential fact that needs to be established here is:

> Within a given feature collection, each feature tag may have only one value.

This idea is explained below in the context of using the media feature framework to describe the characteristics of transmitted image data.

In this context, we have the requirement that any feature tag value must apply to the entire image, and cannot have different values for different parts of an image. This is a consequence of the way that the framework of feature predicates is used to describe different possible images, such as the different images that can be rendered by a given recipient.

This idea is illustrated here using an example of a flawed feature set description based on the TIFF image format defined for use by Internet fax [13]:

```
(& (& (MRC-mode=1) (stripe-size=256) )
   (| (& (image-coding=JBIG-2-LEVEL) (stripe-size=128) )
      (image-coding=[MH,MR,MMR]) ) )
```

This example is revealing because the 'stripe-size' attribute is applied differently to different attributes on an MRC-formatted data: it can be applied to the MRC format as a whole, and it can be applied separately to a JBIG image that may appear as part of the MRC data.

One might imagine that this example describes a stripe size of 256 when applied to the MRC image format, and a separate stripe size of 128 when applied to a JBIG-2-LEVEL coded image within the MRC-formatted data. But it doesn't work that way: the predicates used obey the normal laws of Boolean logic, and would be transformed as follows:

```
--> [flatten nested (&...)]:
    (& (MRC-mode=1) (stripe-size=256)
       (| (& (image-coding=JBIG-2-LEVEL) (stripe-size=128) )
          (image-coding=[MH,MR,MMR]) ) )

--> [Distribute (&...) over (|...)]:
     (| (& (MRC-mode=1) (stripe-size=256)
           (& (image-coding=JBIG-2-LEVEL) (stripe-size=128) ) )
        (& (MRC-mode=1) (stripe-size=[0..256])
           (image-coding=[MH,MR,MMR]) ) )
```

```
   --> [Flatten nested (&...) and group feature tags]:
        (| (& (MRC-mode=1)
              (stripe-size=256)
              (stripe-size=128)
              (image-coding=JBIG-2-LEVEL) )
           (& (MRC-mode=1)
              (stripe-size=256)
              (image-coding=[MH,MR,MMR]) ) )
```

Examination of this final expression shows that it requires both ’ stripe-size=128’ and ’stripe-size=256’ within the same conjunction. This is manifestly false, so the entire conjunction must be false, reducing the entire predicate expression to:

```
        (& (MRC-mode=1)
           (stripe-size=256)
           (image-coding=[MH,MR,MMR]) ) )
```

This indicates that no MRC formatted data containing a JBIG-2-LEVEL coded image is permitted within the feature set, which is not what was intended in this case.

The only way to avoid this in situations when a given characteristic has different constraints in different parts of a resource is to use separate feature tags. In this example, ’MRC-stripe-size’ and ’ JBIG-stripe-size’ could be used to capture the intent:

```
   (& (& (MRC-mode=1) (MRC-stripe-size=256) )
      (| (& (image-coding=JBIG-2-LEVEL) (JBIG-stripe-size=128) )
         (image-coding=[MH,MR,MMR]) ) )
```

which would reduce to:

```
        (| (& (MRC-mode=1)
              (MRC-stripe-size=256)
              (JBIG-stripe-size=128)
              (image-coding=JBIG-2-LEVEL) )
           (& (MRC-mode=1)
              (MRC-stripe-size=256)
              (image-coding=[MH,MR,MMR]) ) )
```

The property of the capability description framework explicated above is captured by the idea of a "feature collection" which (in this context) describes the feature values that apply to a single resource. Within a feature collection, each feature tag may have no more than one value.

The characteristics of an image sender or receiver are described by a "Feature set", which is formally a set of feature collections. Here, the feature set predicate is applied to some image feature collection to determine whether or not it belongs to the set that can be handled by an image receiver.

8. 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 that 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.

9. Acknowledgements

Thanks are due to Larry Masinter for demonstrating the breadth of the media feature issue, and encouraging the development of some early thoughts.

Many of the ideas presented derive from the "Transparent Content Negotiation in HTTP" work of Koen Holtman and Andy Mutz [4].

Early discussions of ideas with the IETF HTTP and FAX working groups led to further useful inputs from Koen Holtman, Ted Hardie and Dan Wing. The debate later moved to the IETF 'conneg' working group, where Al Gilman and Koen Holtman were particularly helpful in refining 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.

10. References

[1] Hardie, T., "Scenarios for the Delivery of Negotiated Content", Work in Progress.

[2] Klyne, G., "Requirements for protocol-independent content negotiation", Work in Progress.

[3] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, March 1999.

[4] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, March 1998.

[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] Masinter, L., Holtman, K., Mutz, A., and D. Wing, "Media Features for Display, Print, and Fax", RFC 2534, March 1999.

[7] Wahl, M., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

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

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

[10] Crocker, D., Editor, and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, 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] "Logic and its Applications", Edmund Burk and Eric Foxley, Prentice Hall, Series in computer science, ISBN 0-13-030263-5, 1996.

[13] McIntyre, L., Buckley, R., Venable, D., Zilles, S., Parsons, G. and J. Rafferty, "File Format for Internet Fax", RFC 2301, March 1998.

[14] Apache content negotiation algorithm, <http://www.apache.org/docs/content-negotiation.html>

11. Author's Address

Graham Klyne
Content Technologies Ltd.
Forum 1
Station Road
Theale
Reading, RG7 4RA
United Kingdom

Phone: +44 118 930 1300
Facsimile: +44 118 930 1301
EMail: GK@ACM.ORG

5th Generation Messaging Ltd.
5 Watlington Street
Nettlebed
Henley-on-Thames
RG9 5AB
United Kingdom.

+44 1491 641 641
+44 1491 641 611

Full Copyright Statement