OpenDMAP Project

MANUAL

Pattern Manual

OpenDMAP Project

Center for Computational Pharmacology

August 2007

1. Introduction to OpenDMAP

OpenDMAP is an ontology-driven, rule-based concept analysis and information extraction system. OpenDMAP elevates the use of conceptually organized semantic information to be the primary organizing architecture of the system. Unlike traditional parsers, OpenDMAP does not have a lexicon that maps from words to all the possible meanings of these words. Rather, each concept is associated with phrasal patterns that are used to recognize that concept. OpenDMAP processes texts to recognize concepts and relationships from a knowledge-base.

OpenDMAP uses Protégé knowledge-bases (http://protege.stanford.edu/) to provide an object model for the possible concepts that might be found in a text. Protégé models concepts as classes that participate in abstraction and packaging hierarchies, and models relationships as class-specific slots.

OpenDMAP is being developed for use in the domain of molecular biology. All the examples provided in this manual are based on molecular biology information extraction tasks. In particular, the examples cover concepts in PROTEIN TRANSPORT and PROTEIN-PROTEIN INTERACTION. However, OpenDMAP is certainly not limited to this domain – it can be applied to any domain for which a knowledge-base exists or can be developed.

There are many examples included in this tutorial. Almost all the example patterns here are from a patterns file and Protégé knowledge-base that comes with your OpenDMAP installation. There may be slight variations between the examples presented here and the patterns and Protégé knowledge-base provided with the installation. Those examples that are not in the files provided with the OpenDMAP installation are noted. A note of caution about the biological content in this manual is in order: the patterns and example sentences illustrated here were made up to best illustrate OpenDMAP's pattern syntax and matching behavior. The knowledge-base accurately reflects the biological world as we have constructed it in the knowledge-base. However, the example sentences have been made up and do not necessarily reflect biological truth. Likewise, the patterns presented here were created to illustrate OpenDMAP's capabilities, but may not show the best way to construct a pattern to extract a particular concept from text.

2. Introduction to the domains used in this manual

Two biological concepts, PROTEIN TRANSPORT and PROTEIN-PROTEIN INTERACTION, are used throughout the pattern manual to illustrate the OpenDMAP pattern syntax.

PROTEIN TRANSPORT is the directed movement of proteins from one cellular compartment to another. The PROTEIN TRANSPORT frame has four slots: what protein is transported, from where, to where, and by what protein. Figure 1 shows the structure of the PROTEIN TRANSPORT concept frame in the Protégé knowledge-base.

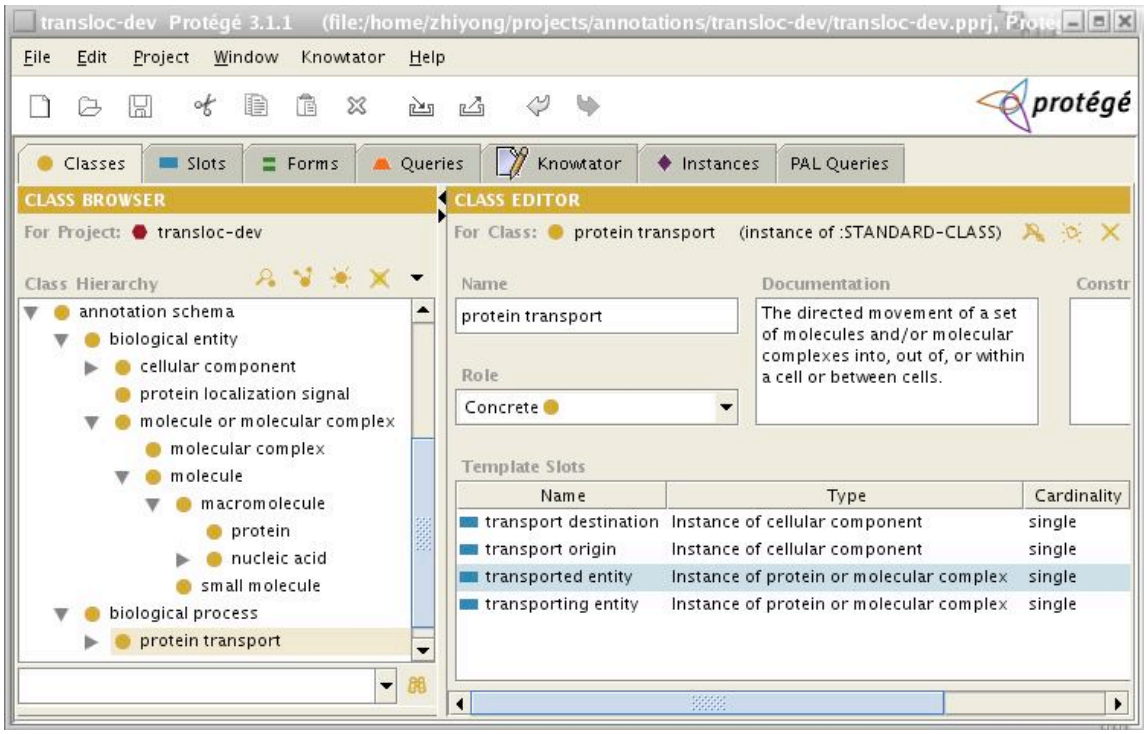

Figure 1: Structure of the PROTEIN TRANSPORT concept frame in the Protégé knowledge-base.

Highlighted at the bottom of the left Class Browser panel is the *class* "protein transport". A Protégé *class* is a set whose members are called *instances*. A class can have any number of *slots*, which are attributes of the *class*. The *slots* of the "protein transport" *class* are shown in the Template Slots field of the Class Editor panel at the right. The names of the four *slots* are shown in the left-most column of that field: "transport destination", "transport origin", "transported entity", and "transporting entity". Possible values are specified for every *slot*; these values constrain what types of entities can be *instances* of a *slot*. A *slot* can have either atomic value constraints (e.g., strings, integers), or it can be specified that any value must be a member of one or more *classes*. In the figure above, the *slots'* allowed values are shown in the middle column of the Template Slots field: *slots* "transport destination" and "transport origin" are constrained to be instances of the *class* "cellular component", and *slots* "transported entity" and "transporting entity" are constrained to be instances of the *classes* "protein" or "molecular complex". Note that the *classes* "cellular component", "protein", and "molecular complex" are part of the same Protégé project, and are present in the Class Hierarchy in the left panel.

PROTEIN-PROTEIN INTERACTION describes the physical interaction of two proteins. Physical interactions often occur between groups of proteins in complexes, but the model of the PROTEIN-PROTEIN INTERACTION concept provided here decomposes multiple protein interactions into pair-wise interactions. This is represented by a PROTEIN-PROTEIN INTERACTION concept frame that has two protein slots. Figure 2 shows the structure of the PROTEIN-PROTEIN INTERACTION concept frame in the Protégé knowledge-base.

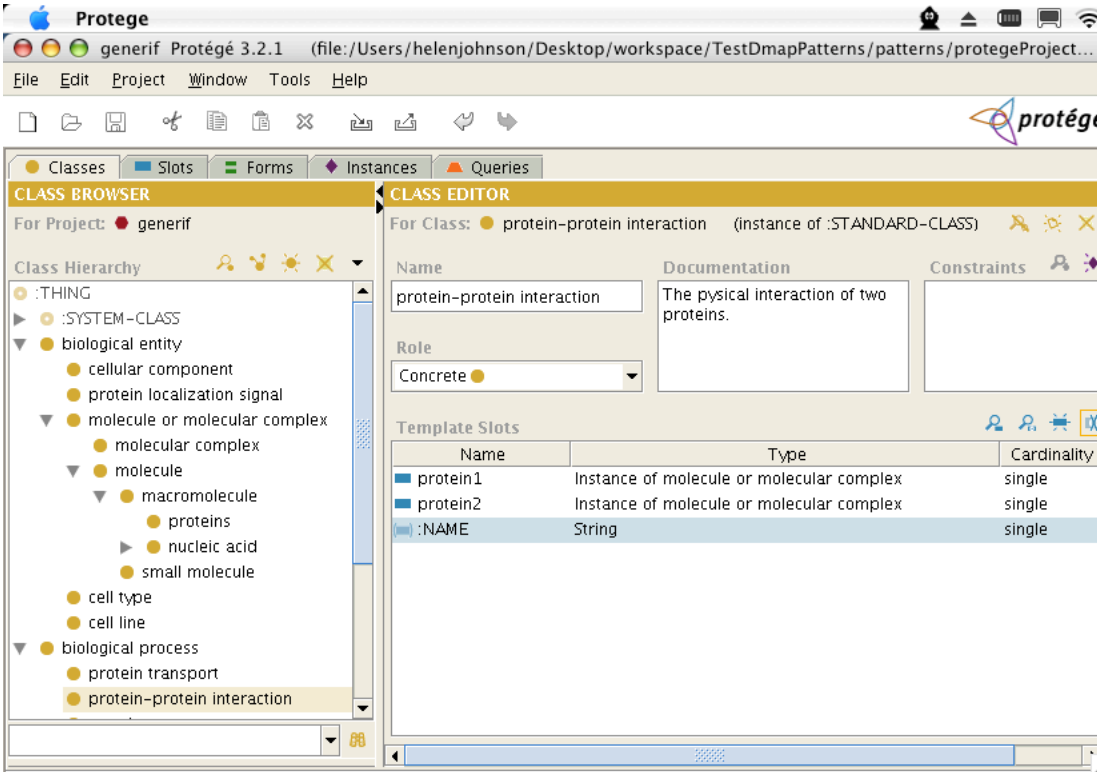

Figure 2: Structure of the PROTEIN-PROTEIN INTERACTION concept frame in the Protégé knowledge-base.

Parallel with the PROTEIN TRANSPORT concept, the "protein-protein interaction" class is shown in the Class Browser panel on the left. It's slots, "protein1" and "protein2" are shown in the Class Editor panel in the Template Slots field. Both slots are constrained to be instances of the "molecule" class or the "molecular complex" class.

3. OpenDMAP pattern elements

An OpenDMAP pattern consists of a left-hand side (LHS) base-class followed by a one or more right-hand side (RHS) pattern elements. The base-class is a concept that you want to find in text. {protein} and {protein-protein interaction} are base-classes in these patterns:

Example 1: Elements of an OpenDMAP pattern: base-classes and pattern elements

LHS BASE-CLASS		RHS PATTERN ELEMENTS
{protein}	:=	"brca1";
{protein-protein interaction}	:=	[protein1] "binds" [protein2];

The pattern elements following the base-class are what OpenDMAP will be looking for in text you input. There can be as few as one pattern element in a pattern, and there is no maximum. Pattern elements can be of several types:

- Strings
- Class references

- Slot references

Regular expressions can be used on strings, class references, slot references, and on combinations of those elements, (see Section 3.5).

Some of these pattern elements stand on their own, meaning they do not refer to outside resources – these elements include strings and regular expressions. The other elements – class references and slot references – are linked elements, meaning they must be linked to an ontology.

3.1. Basic pattern syntax

The OpenDMAP pattern syntax requires that every pattern have, at a minimum, these four elements:

Left-hand side:
- Base-concept: the concept you want to extract from text
- The equals sign :=

Right-hand side:
- Pattern elements (described in Sections 3.2 through 3.4)
- Pattern-final punctuation semicolon ;

OpenDMAP patterns are arranged like:

Example 2: Syntax of an OpenDMAP pattern

```
{protein}          :=          "brca1"          ;
BASE-CONCEPT  EXPANDS-TO  PATTERN-ELEMENTS  SEMI-COLON
```

Pattern elements on the right side of the equation can be combined using a comma to mean 'OR':

Example 3: Concatenating pattern elements with "OR" using commas

Pattern: {protein} := "brca1", "brca-1", "brca 1";
Matches: We show that brca1 plays a role in DNA-repair.
We show that brca-1 plays a role in DNA-repair.
We show that brca 1 plays a role in DNA-repair.

All OpenDMAP syntactic components are listed in Table 1.

Syntax element	Req.	Desc. in Section	Purpose	Usage
:=	yes	3.1	Equates the base-class with its pattern	{protein} **:=** "brca1";
;	yes	3.1	Pattern-final punctuation	{protein} := "brca1"**;**
{class}	yes	3.1/3.3	Designates the base-class in a pattern;	**{protein-protein interaction}** := [protein1] "binds" [protein2];
	no	3.3	Used to reference non-base-classes in patterns	{protein-protein interaction} := [protein1] **{interact-verb}** [protein2];
[slot]	no	3.4	Designates a slot filler element to a frame concept in a pattern	{protein-protein interaction} := **[protein1]** "binds" **[protein2]**;
r' '	no	3.5	Introduces a regular expression	{protein} := **r'**imp[0-9]+**'**;

*	no	3.5	Kleane star – none or more	{protein} := r'imp[0-9]**';
+	no	3.5	Kleane plus – one or more	{protein} := r'imp[0-9]+';
?	no	3.5	Designates an optional pattern element	{protein-interaction} := [protein1] "binds" **with?** [protein2];
_ (underscore)	no	3.6	Wildcard that matches none or more words	{protein-protein interaction} := [protein1] **_** "interacts with" [protein2];
@	no	3.6	Designates a pattern element as optional and floatable	{protein transport} := [transported entity] translocates **@ (from [transport origin])**
" "	no	3.2	Surrounds text strings that will be matched exactly. Using quotes to mark exact text is optional	The following will match the same text: {golgi} := **"**Golgi apparatus**"**; {golgi} := Golgi apparatus;
|	no	3.5	Boolean OR operator	{golgi} := "Golgi apparatus" **|** "Golgi ribbon";
,	no	3.1	Separates pattern element in a list	{interaction-verb} := binds**,** interacts**,** phosphorylates;
( )	no	3.5 3.6	Used in regular expressions Used to group complex elements	{interaction-verb} := interac**(**t|ts|ted|ting**)** {protein transport} := [transported entity] translocates **(**from [transport origin]**)**
//	no		Starts a comment line	//protein patterns follow

Table 1: Syntactic elements available in the OpenDMAP pattern syntax. Most of the elements in this table are not required: column 2 ("Req.") lists which patterns elements are required. In the "Usage" column, text in bold shows the listed element in action.

3.2. String pattern elements

The simplest type of pattern element is a string. A string element is written either as plain text or as text surrounded by double quotes. String elements are not case sensitive. These three patterns will match the same text:

Example 4: Patterns using a string element

Pattern: {protein} := "brca1";
{protein} := brca1;
{protein} := BRCA1;

Matches: We show that BRCA1 exists in a distinct SNARE complex.

String elements are dependent on word boundaries. Word boundaries are marked by whitespace, commas, end of sentence punctuation, and single quotes. String elements do not match partial words in text.

Example 5: String elements do not match partial words.

Pattern: {protein} := brca1;

Matches: BRCA1 exists in a distinct SNARE complex.
BRCA1, GS28 and Ykt6 exist in a distinct SNARE complex.
BRCA1's presence in a distinct SNARE was detected.

Doesn't match: BRCA1a exists in a distinct SNARE complex.
BRCA1-2 exists in a distinct SNARE complex.

The one exception to the optional nature of quotes is when you want to match a string of text that happens to also be a symbol used in a pattern syntax element (see Table 1 for the list of syntax elements). The most common example of this is matching commas in text. A pattern that accounts for a comma in text must look like this:

Example 6: Matching pattern symbols when they appear in text

Pattern: {protein} := brca1“,” 2;
Matches: The proteins brca1, 2 is associated with cancer.

There is a known bug regarding matching commas in text. If the comma in the pattern falls between two digits, with no space after the comma, OpenDMAP will not match the pattern.

Example 7: Known bug in OpenDMAP regarding matching commas in text

Pattern: {protein} := braca1“,”2;
Does not match: brca1,2

3.3. Class reference pattern elements

A class is a concept that has been defined in the Protégé ontology. (Refer back to Section 2 for a brief introduction to Protégé classes, and to the Protégé tutorials at http://protege.stanford.edu for a more in-depth understanding.) So a class reference is a reference to that Protégé class in a pattern. Class references are always in {braces}. Class references must have the same name as the class name in the ontology. The class references cannot have punctuation in their names, and so by extension, neither can the class names in the ontology. They can have white space, underscores and dashes in their names.

Example 8: Using a class reference as a pattern element

Pattern: {protein transport} := [transporter] {transport-verb}[transported];
Matches: Our study found that Imp3 transported BRCA1.

Classes must be conceptually constrained to be useful. Constraining a class means describing the set of things that can belong to the class. The constraints can be placed on classes in two ways:

- In the patterns file
- In the ontology

3.3.1. Constraining classes in the pattern file

Classes can be constrained in the pattern file. This is done by listing strings or other class references that belong to the class you are constraining in the pattern file. Even though the class is constrained by defining the class reference in the pattern file, there still needs to be a corresponding class in the ontology.

Example 9: Constraining a class in the pattern file

Class Pattern:	{interact-verb} := associates, binds, interacts, phosphorylates;
Concept Pattern:	{protein-protein interaction} := [protein1] {interact-verb} [protein2];
Matches:	Our study shows that IMP3 phosphorylates BRCA1.

Example 10: Constraining a class with other class references in the pattern file

Class pattern:	{interact-noun} := interaction;
Class pattern:	{interact-verb} := interact, interacts, interacted;
Class pattern:	{interact-predicate} := {interact-noun}, {interact-verb};
Concept pattern:	{protein-protein interaction} := [protein1]{interact-predicate} with [protein2];
Matches:	Our study showed that Brca1 interacted with imp3.

Constraining classes in the pattern file is a common way of using syntactic type information if you don't have a syntactic tagger in place. For instance:

Example 11: Constraining classes in the pattern file to model syntax

Class Pattern:	{interact-verb} := associates, binds, interacts, phosphorylates;
Class Pattern:	{determiner} := a, the;
Class Pattern:	{preposition} := in, on, to, by, with;
Class Pattern:	{aux-verb} := is, are, was, were;
Concept pattern:	{protein-protein interaction} := [protein1] {aux-verb} {interact-verb} {preposition} {determiner} [protein2];
Matches:	Our study showed that Imp3 is associated with BRCA1.

3.3.2. Constraining classes from within Protégé

Classes can be constrained from within the Protégé ontology. In this case, classes are assigned characteristics that restrict the kinds of entities that can belong to that class. The characteristics are other classes that exist in the ontology. For detailed instructions on how to constrain classes, please refer to the tutorials provided by Protégé at http://protege.stanford.edu/doc/users.html.

3.4. Slot reference pattern elements

As introduced in Section 2, slots define the attributes of a class. For instance, in the case of PROTEIN-PROTEIN INTERACTION, the slots are the proteins that are involved in an interaction; they are labeled "protein1" and "protein2". In the case of PROTEIN TRANSPORT there are four slots: "transported entity", "transporting entity", "transport origin", and "transport destination". Slots must be defined and attached to a class in the ontology. Like classes, slots are only meaningful if they have been conceptually constrained. Slots cannot be constrained in a pattern file – they must be constrained from with Protégé. The method for constraining slots in Protégé was touched on in Section 2.

For a detailed description on how to define and constrain slots, and to assign slots to classes, please refer to the Protégé tutorials provided at http://protege.stanford.edu/.

A slot reference is a reference to a slot in a pattern. Slot references are always written in [brackets]. The name of the slot reference in the pattern must match the slot name in the ontology.

3.4.1. Simple Slot References

Slot references can be simple or complex. Simple slot references contain only the reference to the slot in the ontology. So far in this manual, Examples 8-11 used simple slot references.

To illustrate how OpenDMAP integrates slot information into its matching algorithm, take the following example:

Example 12: Simple slot references in a pattern

Pattern: {protein-protein interaction} := [protein1] binds with [protein2];
Matches: Imp3 binds with BRCA1.

OpenDMAP identifies "Imp3" and "BRCA1" as proteins either because there are explicit OpenDMAP patterns, such as these:

Example 13: Example {protein} class constraints in pattern file

Pattern: {protein} := imp1, imp2, imp3;
Pattern: {protein} := brca1, brca-1, brca 1;

or because "Imp3" and "BRCA1" were tagged as proteins in a preprocessing step involving a gene-tagger. When OpenDMAP identifies these entities as proteins, it creates an instance of a PROTEIN concept for each one. Instances of PROTEINS match the constraints of the [protein1] and [protein2] slots of the PROTEIN-PROTEIN INTERACTION frame, which causes those slots to match to the protein instances associated with the proteins in the text. Then OpenDMAP matches the words "binds" and "with" in text to the "binds" and "with" string elements in the pattern, thereby completing a {protein-protein interaction} concept match. Since the entire pattern matches, an instance of PROTEIN-PROTEIN INTERACTION is created, with the Imp3 protein concept in its [protein1] slot and an instance of the BRCA1 concept in its [protein2] slot.

3.4.2. Complex Slot References

Complex slot references are composed of the slot name plus additional constraints. Those constraints can be syntactic or semantic in nature. For instance, if the text is pre-processed with a dependency parser, an additional syntactic constraint requiring the slot to be in a direct dependence relationship to the verb (i.e. to be the subject, object or modifier of the verb) can be placed after the slot name in the slot reference. The following example illustrates this use. The slot reference [protein1 dep:x] in the example is semantically constrained to be a protein or molecular complex (constrained in the ontology), and syntactically constrained to be a dependent of the verb (constrained by the

addition of the "dep:x" in this pattern, and recognized by the syntactic dependency parser).

Example 14: Complex slot reference using a syntactic dependency constraint

Note: this pattern and text are not included with installation files because they require other resources that are not distributed with OpenDMAP (a dependency parser).

Pattern: {protein-protein interaction} := [protein1 dep:x] _ interacts with [protein2];

Matches: Brca1, a ring finger protein, interacts with imp3.

3.5. Regular Expressions

Regular expressions can be applied to strings, class references, or slot references. There is a difference concerning which regular expression syntax can be used on strings and which can be used on class and slot references, which is detailed below.

3.5.1. Regular expressions over strings

The OpenDMAP regular expression syntax follows Java regular expression syntax. Some regular expressions in patterns must be introduced by the character r and single quotes around the regular expression, as in:

Example 15: Syntax of a regular expression in a string pattern element

Pattern: {protein} := r'imp[0-9]';

Matches: IMP3 is a novel biomarker for adenocarcinoma.

Example 16: Syntax of a regular expression in a string pattern element

Pattern: {interact-verb} := r'associat(e|es|ed)', r'bin(d|ds)', r'interac(t|ts|ted)', r'phosphorylat(e|es|ed)';

Matches: IMP3 interacts with brca1.

Other regular expressions symbols – such as + * ? | – can be used with the r' ' syntax or without. Their behavior differs depending on which context they are used in. Inside the r' ' syntax, those symbols act just like Java-regular expression symbols. Used at the end of strings without the r' ' syntax, they act on the whole word. For instance, notice the different behavior in this silly example:

Example 17: Regular expression symbols with and without the regular expression r ' ' syntax

Pattern: {protein-protein interaction} := [protein1] binds with+ [protein2];

Matches: IMP3 binds with with with GS15.

Doesn't match: IMP3 binds withhh GS15.

Pattern: {protein-protein interaction} := [protein1] binds r'with+' [protein2];

Matches: IMP3 binds withhh GS15.

Doesn't match: IMP3 binds with with with BRCA1.

Example 18: Using the regular expression symbol | with whole words

Pattern: {protein} := mkl1, megakaryoblastic (leukemia-1| (leukemia 1));
Matches: Megakaryoblastic leukemia 1 is a myocardin-related transcription factor.
Megakaryoblastic leukemia-1 is a myocardin-related transcription factor.

3.5.2. Regular expressions over class and slot references

A few of the regular expression syntax elements can be used over class and slot references. These symbols include the following: * + ? |. Class and slot references are described in Sections 3.3 and 3.4, but for now just recognize that class references appear in {braces} and slot references appear in [brackets].

Example 19: Using regular expression syntax on class references

Base pattern: {interact-verb} := phosphorylates, binds, activates;
Base pattern: {interact-noun} := phosphorylation, binding, activation;
Pattern: {protein-protein interaction} := [protein1] ({interact-verb}|{interact-noun) {preposition}? [protein2];
Matches: BRCA1 phosphorylates imp3.
BRCA1 phosphorylation of imp3 is noted in several experiments.

Example 20: Using regular expression syntax over slot references

Pattern: {protein transport} := [transporter] {transport-verb} [transported] (from the [source])? (to the [destination])?
Matches: IMP3 transports BRCA1 from the golgi to the nucleus.
IMP3 transports BRCA1 to the nucleus.
IMP3 transports BRCA1 from the golgi.
IMP3 transports BRCA1.

3.6. Additional OpenDMAP elements: the _ and the @

There are two non-Java regular expression symbols – the underscore _ and the at sign @ – that have been added into the OpenDMAP repertoire. Neither one is used with the r‘ ’ syntax. However, @ elements can contain r‘ ’ elements.

3.6.1. The wildcard symbol _

The underscore is a wildcard symbol that will match zero or more words between other elements in your pattern. In the example below, note in (1) and (2) that OpenDMAP will return two different matches from one text because of the wildcard symbol. Also note in (4) that using the wildcard can hurt your results if not used judiciously.

Example 21: Using the wildcard symbol _ in patterns

Pattern: {protein-protein interaction} := [protein1] _ interact with [protein2];
Matches: (1) Brca1 and IMP3 interact with mk11.
(2) Brca1 and IMP3 interact with mk11.
(3) Brca1 was shown to interact with mk11.
(4) Brca1 does not interact with mk11.

3.6.2. The optional and free-order symbol @

The @ is a symbol that allows you to assign an optional and free-order characteristic to a pattern element. The optional part means the pattern will match text even if the element is not present. The free-order part means the pattern will match even if the element is not in the same order as written in the pattern. More specifically, as long as the element is consecutive (precedes or follows) its 'anchor' in text. An anchor is any other pattern element or series of elements that precedes the @ element.

An @ element cannot begin a pattern; it must follow the 'anchor'. The proper syntax is the @ sign followed by a space, followed by one or more pattern elements. If you forget to put a space between the @ sign and the element, OpenDMAP will run but it will not find any matches with that pattern.

Example 22: Using the optional and free-order simple @ element in patterns

Pattern: {protein-protein interaction} := interacts @ imp3;
Matches: imp3 interacts.

To be precise, the anchor is all elements that precede the @ element. Currently, there is no way to limit the anchor to be just a portion of the elements that precede the @ element. Everything that comes before the @ element is considered the anchor. Similarly, the @ element is understood to be all elements that follow the @ symbol to the end of the pattern, or until another @ symbol is encountered. Like the anchor, there is no way to limit the @ element to a subset of what follows the @ symbol. For clarity, @ elements have been placed in parentheses in these examples, even though they are not necessary.

Example 23: Demonstrating multiple elements in anchor and @ elements

Pattern: {protein-protein interaction} := interacts directly @ (the [protein1]);
Matches: The imp3 interacts directly in vitro.
Interacts directly the imp3 in vitro.
Doesn't match: Interacts the imp3 directly in vitro.

Keep in mind that because the @ element is an optional element, if any part of the @ element is missing from the text, the pattern may still match but it will not include any part of the @ element in its matching results. Reusing the above example in the example below, notice that the determiner "the" that is stated in the @ element in the pattern is missing from the text. OpenDMAP doesn't find the complete @ element in text, but it does find the rest of the pattern, so it returns the matching text that is underlined below.

Example 24: @ elements are optional at pattern matching time

Pattern: {protein-protein interaction} := interacts directly @ (the [protein1]);
Matches: imp3 interacts directly in vitro.

Multiple @ elements can be used in a pattern. As long as the @ elements are found consecutive with each other and the anchor in text, a match will be made. Each @ element is optional, so a pattern will match even if one or more of the @ elements is not present in text.

Example 25: Using multiple @ elements in a single pattern

Note: The sentences in the Matches list are nonsensical, but are there to illustrate OpenDMAP's matching behavior.

Pattern: {protein transport} := {aux-verb} {transport-verb} [transported] @ (from the [source]) @ (to the [destination]);
Matches: Brca1 is transported to the nucleus from the golgi.
Brca1 is transported from the golgi to the nucleus.
From the golgi brca1 is transported to the nucleus.
To the nucleus brca1 is transported from the golgi.
From the golgi to the nucleus brca1 is transported.
To the nucleus from the golgi brca1 is transported.
Brca1 is transported.
Brca1 is transported to the nucleus.
Brca1 is transported from the golgi.
From the golgi brca1 is transported.
To the nucleus brca1 is transported.

To remove the requirement that the @ pattern element be found consecutive with its anchor, place a wildcard _ symbol before the @ element.

Example 26: Using the wildcard symbol _ with @ element together

Pattern: {protein transport} := {transport-verb} [transported] _ @ (from the [source]) _ @ (to the [destination]);
Matches: Brca1 translocates to the nucleus in the absence of NHEJ factors from the golgi.

4. Citing OpenDMAP

Please site the following publication when referring to your use of OpenDMAP:

Hunter, L; Lu, Z; Firby, JR; Baumgartner, WA, Jr.; Johnson, HL; Ogren, PV and Cohen, KB. (submitted) OpenDMAP: An open-source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression.

Please refer to the list of relevant publications at the end of the manual that addresses OpenDMAP performance and other related resources.

5. Bibliography

Baumgartner, W.A; Z. Lu; H.L. Johnson; J.G. Caporaso; J. Paquette; A. Lindemann; E.K. White; O. Medvedeva; K.B. Cohen; L. Hunter; (2007) An integrated approach to concept recognition in biomedical text. In *BiocCreative workshop proceedings.* Madrid, Spain.

Baumgartner, W.A; Z. Lu; H.L. Johnson; J.G. Caporaso; J. Paquette; A. Lindemann; E.K. White; O. Medvedeva; K.B. Cohen; L. Hunter; (submitted) An integrated approach to concept recognition in biomedical text. *Genome Biology BioCreative II Special Issue*

Hunter, L; Lu, Z; Firby, JR; Baumgartner, WA, Jr.; Johnson, HL; Ogren, PV and Cohen, KB. (submitted) OpenDMAP: An open-source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression.

Lu, Zhiyong. Text Mining on GeneRIFs. PhD thesis, Computional Bioscience Program, University of Colorado School of Medicine, CO, USA, 2007.

Noy, N.F., et al., "Protege-2000: an open-source ontology-development and knowledge acquisition environment," *AMIA Annu Symp Proc*. p.953 (2003)