PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 10/13

A Keyword-based Guide to Poirot Stories

Edirlei S. de Lima
Bruno Feijó
Simone D. J. Barbosa
Antonio L. Furtado

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 10/13 ISSN 0103-9741
Editor: Prof. Carlos José Pereira de Lucena July, 2013

A Keyword-based Guide to Poirot Stories

Edirlei S. de Lima
Bruno Feijó
Simone D. J. Barbosa
Antonio L. Furtado

{elima, bfeijo, simone, furtado}@inf.puc-rio.br

Abstract: A system (named KW-GPS) to assist users intent on enjoying Web resources related to a domain-restricted collection of stories is described. Each story is referenced in a virtual library in terms of: (1) the URLs of resources associated with the story, which include but are not limited to plot-summaries, narrative texts and videos; and in terms of (2) keywords of different classes, which serve as a multi-aspect index mechanism. The system was initially applied to Agatha Christie's Poirot detective-stories. Its main feature is a rank-and-show process, that first prompts the end-users to indicate their preferences by choosing from class-organized keyword lists displayed on the screen, after which they are in a position to order the system to activate the desired Web resources. Keywords structured as logical terms, admitting variables as parameters, allow to perform other more laborious processes, involving complex selections and the use of story templates to explore structural similarities. It is shown how story templates, as a representation of narrative motifs, offer a major help towards the composition of new stories. A logic-programming tool was developed to implement the system. A reduced version of the tool runs the basic rank-and-show process in mobile devices, such as tablets and cell-phones. An authoring module is provided to help extending the system to other domains, mainly for entertainment applications.

Keywords: Digital Entertainment, Detective Stories, Web Resources, Logic Programming, Story Templates, Mobile Devices.

Resumo: É descrito um sistema (denominado **KW-GPS**) para servir usuários interessados em divertir-se com recursos disponíveis na Web relacionados com uma coleção de estórias pertencentes a um domínio restrito. Cada estória é referenciada numa biblioteca virtual em termos: (1) das URLs dos recursos pertinentes à estória, os quais incluem mas não se restringem a sumários de enredos, textos narrativos e videos; e em termos (2) de plavras-chave de diferentes classes, funcionando como um índice de aspectos múltiplos. O sistema foi inicialmente aplicado às estórias de detetive protagonizadas por Poirot, de autoria de Agatha Christie. O principal componente do sistema é um processo de ranqueamento e exibição, que começa solicitando aos usuários finais que indiquem suas preferências, escolhendo dentre as palavras-chave contidas em listas organizadas por classe mostradas na tela, depois do que estarão em condições de determinar ao sistema que ative na Web os recursos desejados. Palavras-chave estruturadas como termos de lógica, admitindo variáveis como parâmetros, permitem realizar outros processos mais trabalhosos, envolvendo seleções complexas e o uso de moldes de estórias para explorar semelhanças estruturais. Mostra-se como moldes de estórias, representando motivos narrativos, prestam uma ajuda da maior importância à composição de novas estórias. Uma ferramenta de programação em lógica foi desenvolvida para implementar o sistema. Uma versão reduzida da ferramenta roda o processo básico de ranqueamento e exibição em dispositivos portáteis, tais como "tablets" e telefones celulares. É fornecido um módulo de autoria para ajudar a estender o sistema a outros domínios, sobretudo para aplicações de entretenimento.

Palavras-chave: Entretenimento Digital, Estórias de Detetive, Recursos na Web, Programação em Lógica, Moldes de Estórias, Dispositivos Portáteis.

In charge of publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

Poirot said slowly: "- I intend, Hastings, to be very careful in what I say. Let me put it this way. There is a certain person – **X**. In none of these cases did **X** (apparently) have any motive in doing away with the victim. In one case, as far as I have been able to find out, **X** was actually two hundred miles away when the crime was committed."

Curtain: Poirot's Last Case

1. Introduction

Most recommender systems are based on collaborative filtering algorithms and user models that predict the preferences of consumers from a large database of previous interactions with the system. For a user who is searching for specific aspects of a particular series of entertainment products, however, content-based recommender systems may prove to be more adequate in providing such assistance. Nevertheless, it is quite difficult to find services to access the Web following a knowledge-based strategy of recommendation, which is essential to the digital entertainment industry. On the one hand, consumers may need guidance related to the structure and contents of their favorite series. For instance, how to find a detective story in which someone who hated the victim is the crucial witness? On the other hand, authors of digital entertainment products may want to find some actual help, beyond mere inspiration, to start creating new stories from interesting combinations of existing stories that fit certain characteristics.

In this paper we propose a simple but effective system, called **KW-GPS**, to assist users intent on enjoying Web resources related to a previously located domain-restricted collection of stories. Each story is referenced in a virtual library in terms of the following data: (1) the URLs of Web-residing resources associated with the story, including but not limited to plot-summaries, narrative texts, and videos; and (2) keywords of different classes, which serve as a multi-aspect index mechanism. The system was initially applied to Agatha Christie's Poirot detective-stories [1] – hence its acronym, which stands for **K**ey**W**ord-based **G**uide to **P**oirot **S**tories, also reflecting the authors' effort towards a positioning system (i.e. a **GPS** device) accessory to Web navigation.

The keywords are optionally structured as logical terms, which admit variables as parameters. Moreover this keyword-based specification allows performing other more hardworking processes, involving complex selections and the use of story templates to explore similarity. To whoever has reservations vis-à-vis exposing a general audience to such seemingly abstruse notions, we refer to Poirot himself (cf. epigraph above), who did not hesitate to talk of a not yet identified "X" to his friend Captain Hastings and, through him, to the multitude of his constant readers.

A logic-programming tool was developed to implement the system. The modular structure of the tool caters for the different roles of prospective users. Experts on logic programming may want to revise some of our design decisions and modify parts of the main program, thus playing, like we originally did, the role of *designers*. Experts on the domain on hand, not expected to be skilled programmers (though, even in their case, some knowledge of logic programming is desirable, to be able to formulate logic conditioners), would act as *providers*, searching through the Web for stories and associated resources and choosing appropriate indexing keywords; theirs is the task of producing new domain-specification modules, optionally with the help of the authoring module supplied by the system. Finally a simplified rank-and-show facility that hides the logic formalisms and, especially, an interface for mobile devices were designed for those whose sole interest is to watch the stories, i.e. the *end-users* (henceforward simply *users*).

The rest of the paper is organized as follows. Section 2 describes the **KW-GPS** system, mainly through small examples taken from the domain of Poirot stories. Section 3 elaborates on criteria to formulate keyword repertoires. The mobile device interface is the object of section 4. Section 5 briefly surveys related work. Some concluding remarks are presented in section 6. Appendix A lists a program to execute on desktop or laptop computers the simple rank-and-show process over the module specifying the short `My Poirot 1` domain example, used throughout the paper, which is reproduced in appendix B. The interested reader may wish to copy these two programs into files of type '.pl', and run them with the **SWI-Prolog** interpreter.

2. The KW-GPS Tool

2.1. The rank-and-show facility

The system operates on virtual library data previously located on the Web, pertaining to a given domain such as a collection of Poirot stories. Two main sets of clauses represent, respectively, (1) the numbered *library entries*, giving the title of each story and the URLs of the associated resources; and (2) the *index entries*, consisting of keyword lists

of different classes for each story (numbered as in the library entries). Three clauses precede them, the first to name the keyword classes, and the others to act as *conditioners* to the ranking process, as will be explained later.

Our choice of keyword classes was influenced by a seminal study of Tzvetan Todorov [2], wherein the highly-reputed literary theorist remarked that in a detective-story there actually co-exist two narratives: that of the crime, and that of the investigation. One more class was added, due to Poirot's observation (in *Evil Under the Sun*) that: "Murder springs, nine times out of ten, out of the character and circumstances of the murdered person. Because the victim was the kind of person he or she was, therefore was he or she murdered!". We did *not* include a class about the criminal, because we felt we should leave out whatever might function as a "spoiler", ruining the author's effort to keep the suspense until the end.[1] A small example follows (with the URLs in hyperlink format, for the reader's convenience):

```
/* domain My Poirot 1 */

kw_classes([victim, crime, investigation]).
thresholds(St, [V, C, I], T).
keyword_lists(St, [V, C, I]).


% LIBRARY

lib( 1, 'Evil Under the Sun',
        [plot_summary: 'http://www.imdb.com/title/tt0676148/plotsummary?ref_=tt_ql_5',
         wiki: 'http://en.wikipedia.org/wiki/Evil_Under_the_Sun#Plot',
         video: 'http://www.youtube.com/watch?v=siHNgQPaEnc']).

lib( 2, 'Cards on the Table',
        [plot_summary: 'http://www.imdb.com/title/tt0676143/plotsummary?ref_=tt_ql_5',
         wiki: 'http://en.wikipedia.org/wiki/Cards_on_the_Table#Plot_summary',
         video: 'http://www.youtube.com/watch?v=BH_IVwIiUEw']).

lib( 3, 'The Mysterious Affair at Styles',
        [plot_summary: 'http://www.imdb.com/title/tt0100216/plotsummary?ref_=tt_ql_5',
         wiki: 'http://en.wikipedia.org/wiki/The_Mysterious_Affair_at_Styles#Plot_summary',
         video: 'http://www.youtube.com/watch?v=2K2C_EXuWao',
         full_text: 'http://www.gutenberg.org/files/863/863-h/863-h.htm']).

lib(4, 'The Chocolate Box',
       [plot_summary: 'http://www.imdb.com/title/tt0676169/plotsummary?ref_=tt_ql_5',
        wiki: 'http://en.wikipedia.org/wiki/Poirot''s_Early_Cases#The_Chocolate_Box',
        video: 'http://www.youtube.com/watch?v=Dk0lFG4k6vA']).


% KEYWORDS

kws(1, victim, [gender: female, marital_status: married, occupation: actress,
                character: credulous, swindled_by: 'younger man', age_bracket: 30,
                economic_status: rich]).
kws(1, crime, [action: murder, means: strangulation, place: beach, motive: 'financial gain',
               circumstance: 'holiday season', companion: (lover,'younger man')]).
kws(1, investigation, [clue: 'character of the victim', snag: 'time of death',
                       tactic: 'break self-control']).

kws(2, victim, [gender: male, marital_status: single, occupation: 'art collector',
                character: bizarre, age_bracket: 40, economic_status: rich]).
kws(2, crime, [action: murder, means: stabbing, place: 'drawing room',
               motive: 'avoid accusation', circumstance: 'dinner party']).
kws(2, investigation, [clue: 'bridge scores', tactic: 'deceiving trick']).

kws(3, victim, [gender: female, marital_status: married, age_bracket: 70,
                character: credulous, economic_status: 'large fortune',
                attitude: autocratic,    swindled_by: 'younger man',
                occupation: 'social work']).
kws(3, crime, [action: murder, means: poisoning, place: bedroom, motive: 'financial gain',
               circumstance: 'medical treatment', companion: (someone,'younger man')]).
kws(3, investigation, [clue: 'incriminating letter', tactic: 'expose evidence',
                       snag: 'time of death']).

kws(4, victim, [gender: male, age_bracket: 30, character: evil, occupation: politician,
                economic_status: middle]).
kws(4, crime, [action: execution, means: poisoning, motive: 'moral reasons', place: study,
               circumstance: conversation]).
kws(4, investigation, [clue: chocolates, snag: 'mistaken suspect', tactic: confession]).
```

The ranking process is started by entering the command:

[1] A promise to be broken in section 2.3...

```
:- rank(S).
```

whose single parameter must be a variable to be instantiated at the end with a list of story numbers in decreasing order of total number of hits. The user is then asked, for each keyword class, to choose from the respective set of keywords, which are taken from all the stories in the library and displayed on the screen. For the class `victim`, for example, these are:

```
1:age_bracket:30
2:age_bracket:40
3:age_bracket:70
4:character:autocratic
5:character:bizarre
6:character:credulous
7:character:evil
8:economic_status:large fortune
9:economic_status:middle
10:economic_status:rich
11:gender:female
12:gender:male
13:marital_status:married
14:marital_status:single
15:occupation:actress
16:occupation:art collector
17:occupation:politician
18:occupation:social work
19:swindled_by:younger man
choose for victim:
```

To choose, the user types the corresponding numbers, e.g. `7, 12` (for `character:evil` and `gender:male`), and presses the enter key. If the class is not at the moment an aspect of interest, the user simply presses the key without supplying any numbers. Hits are added-up for stories that possess the chosen keywords, but stories without the keywords are not excluded. However the user has the option to prefix a number with either a '+' or a '-' sign, to indicate, respectively, that only stories *with* that keyword or only stories *without* it are acceptable. If the user types any number outside the range the system repeats the question: "please, only numbers between 1 and 19 - choose:". At the end, the output parameter variable is instantiated as mentioned before, and, in addition, the result of the ranking process is shown in sentential form. With the choices `7, 12` for `victim`, `3, -11, 14` for `crime`, and `1, 2, 8` for `investigation`, the result would be:

```
The Chocolate Box with 5 hits
Cards on the Table with 2 hits
The Mysterious Affair at Styles with 0 hits

S = [4, 2, 3]
```

Notice that `-11`, referring to `means:strangulation`, caused the exclusion of *Evil Under the Sun*. Also notice that a story with `0` hits was kept, which is in general useless. To remedy this inconvenience, the `thresholds` clause, shown before in a, so to speak, "neutral" format, can be rewritten in order to impose conditions, both on the total of hits and on the number of hits per class. For instance

```
thresholds(St,[Nvictim,Ncrime,Ninvestigation],T) :-
      Nvictim > 0.
```

would require at least one hit for the class `victim`, and hence a non-zero total as well.

The purpose of the other conditioning clause, `keyword_lists`, is to act as a *filter*, typically considering the permissible user's choices in a general context. For example, it can specify that if the user explicitly rejects stories with `means:stabbing`, then the even more gruesome stories with `means:strangulation` will also be excluded:

```
keyword_lists(St,[Kvictim,Kcrime,Kinvestigation]) :-
  (member( -(means:stabbing), Kcrime),
   kws(St,crime, Ks), member(means:strangulation, Ks), !, fail;
   true).
```

To activate the resources provided for their best-ranked stories, users have a `show` command, whose parameters designate the resource and the story-number. The line below will explore whatever resources are available for *The Chocolate Box*, a story in which Poirot, in his own opinion, acted with less than his usual ingenuity:

```
:- has_resources(St, 'The Chocolate Box', R),
   forall(member(Ri, R), show(Ri, St)).
```

all of which can be watched simultaneously, if non-overlapping windows are adequately disposed on the screen.

2.2. Other basic facilities

The `similar` command uses the keywords of a story to rank the others, consequently giving a measure of how close they are to it. Applying this command to *Evil Under the Sun* we learn that it has something in common with all the other stories, in special with *The Mysterious Affair at Styles*:

```
?- similar(1,S).

The Mysterious Affair at Styles with 7 hits
Cards on the Table with 2 hits
The Chocolate Box with 1 hits

S = [3, 2, 4].
```

Users with a knowledge of the domain specification have available a flexible `select` command to rank the stories on the basis of explicitly indicated keywords, covering one or more classes. For instance, the same result of the example of the preceding section would be obtained by entering the line:

```
?- select([[character:evil, gender:male], [circumstance:'dinner party', -means:strangulation,
            motive:'moral reasons'], [clue:'bridge scores', clue:chocolates,tactic:confession]], S).
```

Of course typing errors should be expected, which led us to introduce a checking device that performs a preliminary comparison of the indicated keywords against those figuring in the `kws` clauses. Automatic substitution will occur if one is found within a *Levenshtein distance* [3] less or equal to 2 from the misspelled keyword. In all such cases a message is displayed, such as:

```
cicunstance:dinner party not found for class {crime} - similar: circumstance:dinner party
```

Automatic substitution will also happen in order to accommodate a number of related terms not included in the `kws` clauses, which, nevertheless, would spontaneously occur to persons familiar with the domain. Indeed we realized that cluttering the `kws` clauses with hypernyms, hyponyms and other *related terms* would affect the intended user-friendliness of the `rank` command in a negative way, since most people would not like to choose from excessively long lists. A compromise solution was adopted, consisting of the addition to the domain specification of specific (as well as somewhat more general) `rel_kw` clauses such as:

```
rel_kw(means:poisoning, means:arsenic).
rel_kw(K, K_rel) :- is_a(K_rel, K).
```

assuming that, to enable the second clause, appropriate `is_a` clauses are also provided, such as

```
is_a(policeman,investigator).
is_a(inspector,investigator).
is_a(detective,investigator).
is_a('large fortune',rich).
```

If the system can neither handle the user's indicated term as a misspelled or as a related reference to a registered keyword, the intractable term is not used in the selection and a warning message is issued:

```
??? pet:Turkish Angora cat not found for class {victim} - dropped from list
```

Yet the most significant feature of the select command is its ability to deal with variables, optionally referred to in logical expressions following the '/' separator. The example below searches for stories with victims of both genders younger than 50. Whenever variables are involved, the selection list is displayed to reveal how they were instantiated upon the execution of the command, thereby performing a complementary query-answering task:

```
?- select([[age_bracket:A, gender:G], [], []]/(A < 50), S).

1 - [[age_bracket:30, gender:female], [], []]
2 - [[age_bracket:40, gender:male], [], []]
4 - [[age_bracket:30, gender:male], [], []]

The Chocolate Box with 2 hits
Cards on the Table with 2 hits
Evil Under the Sun with 2 hits

S = [4, 2, 1].
```

In view of a thesis cogently exposed by George P. Lakoff [4] about categorization, we may consider that the two commands in this section, `select` and `similar`, complement each other in a nice way, given that human beings tend to classify things by just wondering to what other (prototypical) thing they resemble, rarely trying to verify systematically whether they have the properties postulated in scholarly taxonomies. Thus people would promptly

categorize an animal as a bird if it looks like a robin, a bird *par excellence* according to that author. Applying the notion to stories, such users instead of asking: "give me a story with such and such characteristics" (explicit keywords), would say: "give me a story like that one".

2.3. Story templates and narrative motifs

At the beginning of the previous section we saw a case where similarity, as measured in terms of hits with the specified keywords, seemed to lie beyond the limits of pure coincidence. In fact it is possible to detect a common *narrative motif* in *Evil Under the Sun* and *The Mysterious Affair at Styles*, which we shall call the `Swindler motif`. The same motif occurs in several other Poirot stories, the most striking case happening in *Death in the Nile*.

Besides the `Swindler motif`, we shall treat a second motif, to be called the `Inducer motif`. Even though it takes us to the far-removed domain of Elizabethan drama, it will be considered here because Poirot himself explicitly brought it in to find the single responsible for a series of crimes in his last case (reported in the *Curtain* story). Several apparently unrelated criminal offences had been committed by different individuals, but the entire set of events had one thing in common: the presence of a person, whom Poirot simply named "X", who had been in close contact with each of the accused. The little Belgian detective solved the mystery and identified "X" through an analogy with Shakespeare's *Othello*.

To expand our library to accomodate literary motifs, a clausal representation compatible with the `lib` and `kws` story clauses is needed. We might have already pointed out that keyword classes composed of property:value pairs, such as we have been using in the examples, can be characterized as *frames*, a data structure that lends itself well to a method for describing and handling narrative motifs.

Before detailing the method, let us further elaborate the `kws` clauses of *Evil Under the Sun* and *The Mysterious Affair at Styles*, in order to take full advantage of the introduction of story motifs. The novelties are the assignment of proper nouns to some of the main characters, and the addition of terms for properties `culprit` and `accomplice` to the `kws` clauses of class `investigation`. Finding the `culprit` is clearly the main purpose of investigating the crime, whereas recognizing that someone acted as an `accomplice` is sometimes the only way to unmask a suspicious alibi, in this case supported by the `snag` on the whereabout of the prime suspect at the time of the murder. As a matter of fact, in *Evil Under the Sun*, Poirot may have begun to realize that an `accomplice` must have been involved when he learned of a previous unsolved crime (story similarity once again, narrated inside the main story) that the police attributed to a certain man, whom they failed to convict due to the false testimony of a young lady.

The value components of both properties are denoted by variables, but later in this section `X` and `Y` will be instantiated, thereby breaking the customary injunction against spoilers...

```
kws(1, victim, [gender: female, marital_status: married, occupation: actress,
                character: credulous, swindled_by: 'Redfern', age_bracket: 30,
                  economic_status: rich]).
kws(1, crime, [action: murder, means: strangulation, place: beach, motive: 'financial gain',
               circumstance: 'holiday season', companion: ('Christine', 'Redfern')]).
kws(1, investigation, [clue: 'character of the victim',  snag: 'time of death',
                       tactic: 'break self-control', culprit: X, accomplice: Y]).

kws(3, victim, [gender: female, marital_status: married, age_bracket: 70,
                character: credulous, economic_status: 'large fortune',
                attitude: autocratic,    swindled_by: 'Inglethorp',
                occupation: 'social work']).
kws(3, crime, [action: murder, means: poisoning, place: bedroom, motive: 'financial gain',
               circumstance: 'medical treatment', companion: (someone,'Inglethorp')]).
kws(3, investigation, [clue: 'chemical property', tactic: 'expose evidence',
                       snag: 'time of death', culprit: X, accomplice: Y]).
```

Along the same lines, we now add the complete specification of the *Curtain* and *Othello* stories:

```
lib( 5, 'Curtain',
        [wiki: 'https://en.wikipedia.org/wiki/Curtain_(novel)#Plot_summary']).

lib( 6, 'Othello',
        [plot_summary: 'http://en.wikipedia.org/wiki/Othello#Plot',
         video: 'http://www.youtube.com/watch?v=fETm6neCEJ0&list=PL48F22AF61832B590',
         opera: 'http://www.youtube.com/watch?v=_BdTrzHEnYs',
         full_text: 'http://www.gutenberg.org/cache/epub/1127/pg1127.html']).

kws(5, victim, [victim_name:'Desdemona']).
kws(5, crime, [loves:('Othello','Desdemona'),tells:('Iago','Othello',infidel('Desdemona')),
               kills:('Othello','Desdemona'),suicides: 'Othello']).
kws(5, investigation, [culprit:X]).

kws(6, victim, [victim_name:wife]).
kws(6, crime, [tells:('Norton', 'Riggs', infidel(wife)), loves:('Riggs',wife),
```

```
                kills:('Riggs',wife)]).
kws(6, investigation, [culprit:X, executes:('Poirot',X)]).
```

Our method to specify and then to handle frame-structured *story templates*, to be added to the library as a representation of narratif motifs, employs two operations: *unification* [5] and its dual, *most specific generalization* (**msg** for short). Let us recall how each operation, starting with **msg**, works on frames.[2] If `F1` and `F2` are frames, their **frame-msg** `Fm` is obtained as follows. Only property:value pairs referring to the same property `p`, say `p:v1` from `F1` and `p:v2` from `F2` are considered, yielding as part of the resulting `Fm` the pair `p:vm`, where `vm` is in turn the **msg** of the values `v1` and `v2`. For value components that are both constants, if they are equal, their **msg** is this value itself, otherwise it is a variable. A variable also results if one or both value components are variable. An important extension applies for constants in the presence of `is_a` links: the resulting value is the more general term. So, for the `economic_status` property, the **msg** of `'large fortune'` and `'rich'` is `'rich'`.

With the help of **msg**, it is possible to combine the common features of two or more stories so as to create story templates able to represent a recurring motif. For detective stories it is expedient to start with the narrative of the crime. Accordingly, when considering two apparently similar stories, one first computes the **frame-msg** of their `kws` clauses of class `crime`, in order to obtain the `kws_t` clauses of class `crime` that will be part of the story template being constructed. In the current implementation, we decided, as a simplification, to drop the property:value pairs whose value components came up as variables (therefore not imposing any specific value for the common property). To obtain the `kws_t` clauses for the two other classes, `victim` and `investigation`, we equally begin by computing the **frame-msg** of the respective `kws` clauses of the stories. But, next, we must add to these `kws_t` clauses certain *connection terms* to establish a link with the crime `class` previously specified. Intuitively, the connection terms in the first motif are needed to infer, in the course of the investigation, that whoever plays the role of swindler is the `culprit` and his companion the `accomplice`. For the second motif, the inference attributes to the perpetrator of the inducement full responsibility as the true `culprit` of the crime.

Following these steps, we proceed to add to our library story templates representing the two motifs, with their `lib_t` and `ksw_t` clauses fully format-compatible with the `lib` and `ksw` clauses of the stories:

```
lib_t( 1, 'Swindler motif',[]).

kws_t(1, victim, [gender: female, character: credulous, marital_status:married,
                  economic_status: rich, swindled_by: X]).
kws_t(1, crime, [action: murder, motive: 'financial gain', companion: (Y,X)]).
kws_t(1, investigation, [snag: 'time of death', swindled_by: X, companion: (Y, X),
                         culprit: X, accomplice: Y]).

lib_t(2, 'Inducer motif',[]).

kws_t(2, victim, [kills: (B, A), victim_name:A]).
kws_t(2, crime, [kills: (B, A), loves: (B, A), tells: (C, B, infidel(A))]).
kws_t(2, investigation, [tells: (C, B, infidel(A)),
                         culprit: C]).
```

The two commands that handle this expanded library utilize *frame-unification*, the dual of frame-msg. If `F1` and `F2` are frames, the frame `Fu` resulting from their frame-unification retains all of their property:value pairs, common or not. For each common property, say `p`, if `p:vu` denotes the result of the unification of `p:v1` and `p:v2`, the value `vu` is obtained by unifying `v1` and `v2` as follows. In case both are constant, the result is their value if they are equal, else the unification fails (and in consequence the unification of the two frames also fails). If at least one value component is a variable, the process always succeeds, returning a variable if both are variables, or the one constant value component otherwise. For `is_a` hierarchies, if `v1 is_a v2`, `vu` will be `v1`, the concept with the narrower scope (cf. the unification of *feature structures* in [5]).

The command `similar_t(T,S)` finds which stories incorporate a given motif. Its input parameter is the identifying number of the story template representing the motif, and the output parameter yields the list of stories detected. Informally speaking, what the command does is the inverse of the template-generating process: instead of generalizing and introducing variables when needed, it *specializes* by instantiating variables with constants according to a pattern-matching discipline. Specifically, the command initially applies frame-unification to combine in a single frame `Ft` the three classes of `kws_t` clauses of the the story template. Next, for each story `Si`, a single frame `Fsi` is equally obtained by doing the same with the respective `kws` clauses, and after that `Fsi` is unified with `Ft`. The final unification step, putting together, frame Ft taken from the story template and the Fsi frame of each story, causes, through the propagation effect enabled by the connection terms placed in the story template, the instantiation of the variables with the constants present in the story.

Thus, in our detective stories domain, the `similar_t` command, while verifying whether or not the given motif occurs in each story kept in the library, as a side-effect reveals the names that the investigator was searching for; in

[2] http://www-di.inf.puc-rio.br/~furtado/msg.pdf

other words: it simulates the reasoning practice adopted in reality as much as in fiction, in the world of crime and in other domains, of solving a new problem through a comparison with analogous problems registered in the past. In particular, the execution of

```
:- similar_t(1,S).
```

denounces the criminals of the two stories featuring the `Swindler motif` (except that, as often happens, Poirot does not immediately disclose who is the "someone" in the second story responsible for the `snag` involving the time of death, which served as an alibi to Mr. Inglethorp):[3]

```
Evil Under the Sun - [[gender:female, marital_status:married, occupation:actress, character:credulous,
                       swindled_by:Redfern, age_bracket:30, economic_status:rich],
                      [action:murder, means:strangulation, place:beach, motive:financial gain,
                       circumstance:holiday season, companion: (Christine, Redfern)],
                      [clue:character of the victim, snag:time of death, tactic:break self-control,
                       culprit:Redfern, accomplice:Christine]]

The Mysterious Affair at Styles - [[gender:female, marital_status:married, age_bracket:70,
                                    character:credulous, economic_status:large fortune,
                                    attitude:autocratic, swindled_by:Inglethorp, occupation:social work],
                                   [action:murder, means:poisoning, place:bedroom, motive:financial gain,
                                    circumstance:medical treatment, companion: (someone, Inglethorp)],
                                   [clue:chemical property, tactic:expose evidence, snag:time of death,
                                    culprit:Inglethorp, accomplice:someone]]

S = [1, 3].
```

and, turning to the `Inducer motif`, the command yields:

```
?- similar_t(2,S).

Curtain - [[victim_name:wife],
           [tells: (Norton, Riggs, infidel(wife)), loves: (Riggs, wife), kills: (Riggs, wife)],
           [culprit:Norton, executes: (Poirot, Norton)]]

Othello - [[victim_name:Desdemona],
           [loves: (Othello, Desdemona), tells: (Iago, Othello, infidel(Desdemona)),
            kills: (Othello,Desdemona), suicides:Othello],
           [culprit:Iago]]

S = [5, 6].
```

Story templates, as a representation of literary motifs, also serve a more ambitious goal: to help compose new stories. For this purpose, the `select_t` command is used, having as input parameter, like the `select` command of section 2.2, explicit keyword lists covering all classes (though for some of them empty lists can be supplied). The command uses these terms to instantiate the `kws_t` clauses of the story templates, its output parameter indicating which story templates were successfully matched. Since the command again employs frame-unification to achieve instantiation, properties not in common to the two operands, i.e. that do not figure either in the input lists or in the story template, are kept – so that the story beginning to emerge prolongs, so to speak, the narrative beyond the motif expressed by the story template.

The example introduces two characters unknown in the Poirot world, a man called Archie and his companion Miss Neele; it also adds the novel circumstance that the victim of the (unspecified) criminal action has disappeared. Again by virtue of the first story template, a `culprit` and an `accomplice` are revealed:

```
?-select_t([[swindled_by:'Archie'],
            [companion:('Miss Neele','Archie'),circumstance:'victim disappears'],
            []],S).

Swindler motif - [[swindled_by:Archie, gender:female,
                   character:credulous, economic_status:rich],
                  [companion: (Miss Neele, Archie),
                   circumstance:victim disappears,
                   action:murder, motive:financial gain],
                  [snag:time of death, swindled_by:Archie,
                  companion: (Miss Neele, Archie),
                  culprit:Archie, accomplice:Miss Neele]]

S = [1].
```

Suppose we repeat the command with one more property:value pair. The man's companion performs another evil act: she accuses his wife, called Teresa, of infidelity, thereby reinforcing his murderous impulse. As a result, both story templates are separately instantiated:

[3] If the reader insists to learn, the "someone" is a certain Miss Howard.

```
?- select_t([[swindled_by:'Archie'],
             [companion:('Miss Neele','Archie'),
              tells:('Miss Neele','Archie',infidel('Teresa')),
              circumstance:'victim disappears'],[]],S).

Swindler motif - [[swindled_by:Archie, gender:female,
                   character:credulous, economic_status:rich],
                  [companion: (Miss Neele, Archie),
                   tells: (Miss Neele, Archie, infidel(Teresa)),
                   circumstance:victim disappears, action:murder,
                   motive:financial gain],
                  [snag:time of death,
                   swindled_by:Archie, companion: (Miss Neele,
                   Archie), culprit:Archie, accomplice:Miss Neele]]

Inducer motif - [[swindled_by:Archie, victim_name:Teresa,
                  loves: (Archie, Teresa)],
                 [companion: (Miss Neele, Archie),
                  tells: (Miss Neele, Archie, infidel(Teresa)),
                  circumstance:victim disappears, loves: (Archie,
                  Teresa), kills: (Archie, Teresa)],
                 [tells: (Miss Neele, Archie, infidel(Teresa)),
                  culprit:Miss Neele]]

S = [1, 2].
```

Could a story combining both motifs be composed? The trouble is that putting the two lines together would be rejected by the frame-unification discipline: a conflict arises regarding the identity of the main culprit. Solving conflicts, in order to be able to combine narrative lines, is a task that often requires much creativity; under the name of blending, it is extensively discussed in [6]. As a first rather naive way to face the problem, we introduced a `select_blend` command, which simply asks the user to make a choice wherever a property:value conflict arises:

```
?- select_blend([[swindled_by:'Archie'],
                 [companion:('Miss Neele','Archie'), tells:('Miss Neele','Archie',infidel('Teresa')),
                  circumstance:'victim disappears'],
                 []],S).

conflict with culprit:
1. Archie
2. Miss Neele
Your choice: 2

victim:
[swindled_by:Archie, gender:female, character:credulous, economic_status:rich, victim_name:Teresa,
 loves: (Archie, Teresa)]

crime:
[companion: (Miss Neele, Archie), tells: (Miss Neele, Archie, infidel(Teresa)),
 circumstance:victim disappears, action:murder, motive:financial gain, loves: (Archie,Teresa),
 kills: (Archie, Teresa)]

investigation:
[snag:time of death, swindled_by:Archie, companion: (Miss Neele, Archie), accomplice:Miss Neele,
 tells: (Miss Neele, Archie, infidel(Teresa)), culprit:Miss Neele]
```

Of course a general treatment of conflicts should not stop at this point, since in its fullest sense blending requires rendering the story consistent, which may necessitate several adjustments. Archie murders his wife for monetary gain while still loving her and feeling jealous for a supposed betrayal. Mixed feelings are not uncommon, strange as they may seem, but if Miss Neele is deemed the main `culprit`, would it make sense to also qualify her as `accomplice`? Nonetheless we claim that the example illustrates the notion that new stories can arise by combining two or more motifs, and extending the combination with further events. And, although this little story was surely never composed by Agatha Christie, we suspect that some such plot might well have crossed her imagination. (Why? – we leave that to the reader, as a Web-searching exercise).

2.4. A simple rank-and-show user interface

A reduced version of the **KW-GPS** tool, containing only the `rank` and the `show` facilities, is activated by double-clicking over a shortcut, which may be conveniently placed on the user's desktop area. The user does not have to enter any command line, every move being menu-directed. The keyword lists are successively presented for each class, wherefrom the user chooses as described in section 2.1.

Assuming that the choices were the same as in that example, the same stories are ranked in the decreasing order of total hits, followed by a request to indicate what resource should be activated. Suppose that *Cards on the Table* is

chosen, and the user asks for its plot-summary. While the plot-summary remains open, other resources can be solicited and can be visualized side-by-side if non-overlapping windows are adequately disposed (Figure 1).

When `end` (either the word itself or the corresponding number in the menu) is entered to finish the demand of resources, the list of stories is displayed again. Suppose *The Chocolate Box* is then selected and, from its resources, the Wikipedia entry is called for. Next, if the user twice replies `end` (to the choice of resources and to the choice of stories), these two recursive loops terminate, and the system backtracks to the outermost loop, asking whether the user wants to perform another selection, thereby starting again the whole process. If the answer is negative, a `halt` command is executed and the **SWI-Prolog** window vanishes from the screen.

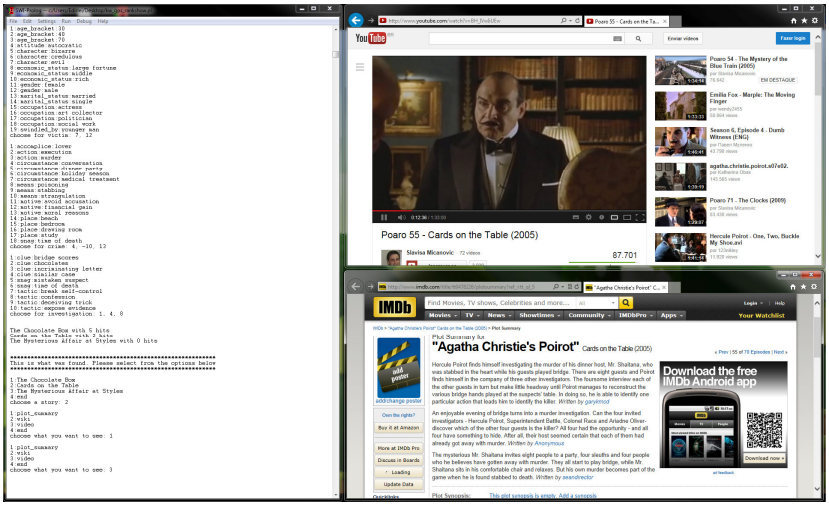

Fig. 1. The simple rank-and-show user interface.

The successive steps of the example session, immediately after the choice of keywords, follow below:

```
The Chocolate Box with 4 hits
Cards on the Table with 3 hits

************************************************************
This is what was found. Please select from the options below
************************************************************

1:The Chocolate Box
2:Cards on the Table
3:end
choose a story: 2

1:plot_summary
2:wiki
3:video
4:end
choose what you want to see: 1

1:plot_summary
2:wiki
3:video
4:end
choose what you want to see: 3

1:plot_summary
2:wiki
3:video
4:end
choose what you want to see: 4

1:The Chocolate Box
2:Cards on the Table
3:end
```

```
choose a story: 1

1:plot_summary
2:video
3:wiki
4:end
choose what you want to see: 3

1:plot_summary
2:video
3:wiki
4:end
choose what you want to see: 4

1:The Chocolate Box
2:Cards on the Table
3:end
choose a story: 4

do you want to try another selection? (y/n):
```

Thanks to this strictly menu-driven usage mode, the programming language formalisms stay hidden, so that the Prolog machinery becomes practically transparent to users. The only hopefully not too serious nuisance is the need to choose by typing, which is obviated in the interface for mobile devices, to be described in section 4.

2.5. Installing a new domain

To create or redesign a domain, providers have an authoring tool that requires only a minimum of familiarity with the notational details of Prolog. The first step is to specify how the domain will be called, the names of the classes of keywords, and information about the stories that will constitute the library. The entries for each story should be numbered consecutively, with the indication of the titles and of the URLs of the resources. In its present implementation, the tool does not provide a general mechanism to retrieve these URLs, which must have been previously found by the provider using some Web search system; however, for certain domains, there may be regularities in the formation of URLs, which may help to partially automate even that task, as will be indicated later in this section for our domain of Poirot stories.

This specification should be written on a text file, entitled `spec.txt`, to be stored in the same folder where the Prolog programs reside. What follows is exactly what is needed to start preparing the example of section 2.1.

```
domain: My Poirot 1

classes:  victim, crime, investigation

1. Evil under the Sun
plot summary: http://www.imdb.com/title/tt0676148/plotsummary?ref_=tt_ql_5
wiki: http://en.wikipedia.org/wiki/Evil_Under_the_Sun#Plot
video: http://www.youtube.com/watch?v=siHNgQPaEnc

2. Cards on the Table
plot summary: http://www.imdb.com/title/tt0478226/plotsummary?ref_=tt_ql_5
wiki: http://en.wikipedia.org/wiki/Cards_on_the_Table#Plot_summary
video: http://www.youtube.com/watch?v=BH_IVwIiUEw

3. The Mysterious Affair at Styles
plot_summary: http://www.imdb.com/title/tt0100216/plotsummary?ref_=tt_ql_5
video: http://www.youtube.com/watch?v=2K2C_EXuWao
wiki: http://en.wikipedia.org/wiki/The_Mysterious_Affair_at_Styles#Plot_summary
full_text: http://www.gutenberg.org/files/863/863-h/863-h.htm

4. The Chocolate Box
plot_summary: http://www.imdb.com/title/tt0676169/plotsummary?ref_=tt_ql_5
video: http://www.youtube.com/watch?v=Dk0lFG4k6vA
wiki: http://en.wikipedia.org/wiki/Poirot's_Early_Cases#The_Chocolate_Box
```

When the provider double-clicks on a shortcut giving access to the `kws_create_domain.pl` program, which may have been placed on the user's desktop area, the following dialogue ensues:

```
Which option do you choose?
1. create domain without keywords?
2. create domain with keywords from IMDB plot-summaries?
Your choice:
```

Both options cause the system to open the `spec.txt` file, from which it extracts the information necessary for creating all clauses, with the major exception of the `kws` clauses, and registering them on a newly created (or again initialized) Prolog file whose name is derived from the name assigned to the domain:

```
file my_poirot_1.pl in use for domain My Poirot 1
```

The choice of the first option implies that the keywords to be used for constructing the `kws` clauses will be later introduced manually by the provider, who should be expected to possess a fair knowledge of the domain, possibly refreshed by skimming through the available summaries.

The second option starts an automatic search over the plot-summaries, employing a keyword-extraction service supplied by **Yahoo**.[4] Each keyword extracted is submitted on a menu-directed fashion to the provider, who either accepts it, by indicating to which class it should be assigned, or rejects it. In case of doubt about the meaning of a candidate keyword, the provider can, before replying, type a question mark (`?`), thereby causing the tool to fetch an explanatory **Wordnet** or **Dbpedia** page (the latter being utilized in case of a capital initial letter).

When keywords have been chosen and duly classified for all stories, and the resulting `kws` clauses stored in the Prolog file assigned to the domain, the system poses a last question to the provider:

```
Domain created and saved on file. Do you want
1. to quit?
2. or to stay online?
Your choice:
```

With the first option a `halt` command is executed and the session is terminated. With the second, the system informs:

```
Domain available for use
```

meaning that the provider already has a chance to test the newly created domain, for which all commands described in the previous sub-sections are available.

Coming back to the preliminary task of finding the URLs of the pages from which the system will extract keywords, we have observed that, for 54 out of the 66 Poirot movies with David Suchet in the leading role, the URLs of the respective **IMDB** plot-summaries form a consecutive sequence, which can be generated by assigning to the two-digits position marked below as "**XX**" numbers in the interval <41-94>:

```
http://www.imdb.com/title/tt06761XX/plotsummary?ref_=tt_ql_5
```

As an alternative, we also considered the extraction of keywords from the summaries supplied in **Wikipedia** pages referring to the original stories narrated in Agatha Christie's books. When the title of a story coincides with the title of the book, the URLs are formed by inserting the title, with all blanks filled in with underscores, in the variable-length substring marked by "Title":

```
http://en.wikipedia.org/wiki/Title#Plot
```

whereas, for the short stories, the title of the collection (e.g. *Poirot's Early Cases*) containing the story should be used instead.

For retrieving the keywords respectively from **IMDB** and from **Wikipedia**, using the **Yahoo** service, we provided the commands:

```
:- imdb_kws(<input: st-number>, <output: list of keywords>).
:- wiki_kws(<input: st-number>, <output: list of keywords>).
```

and, in addition, if the provider wants to verify the occurrence in the **IMDB** plot-summaries of all stories in the current library of a certain word or phrase, no matter if included or not among those found via **Yahoo**, the command `check_term` can be used. Besides the story numbers, the command returns and also prints out other useful information. For instance, some readers might call for a keyword class that would allow them to find stories having the co-participation of characters figuring in the recurring cast of Poirot stories, such as Captain Hastings, Miss Lemon, George, Inspector Japp, and Ariadne Oliver. Here is the result of looking for the Scotland Yard chief inspector:

[4] http://developer.yahoo.com/search/content/V1/termExtraction.html

```
?- check_term('Inspector Japp',S).

Term -- Inspector Japp:
 - in Plot Summary of Evil Under the Sun
     number of times: 0
     tf-idf: 0.0
 - in Plot Summary of Cards on the Table
     number of times: 0
     tf-idf: 0.0
 - in Plot Summary of The Mysterious Affair at Styles
     number of times: 1
     tf-idf: 0.693147
 - in Plot Summary of The Chocolate Box
     number of times: 1
     tf-idf: 0.693147

S = [[1, 0, 0.0], [2, 0, 0.0], [3, 1, 0.693147], [4, 1, 0.693147]].
```

noting that the weighting scheme known as **tf-idf** (term frequency–inverse document frequency) [7] furnishes a widely used criterion to measure the effectiveness of keywords.

But some caveats are in order here. Obviously the term frequency, which is one of the two factors influencing **tf-idf**, is a strong indicator of relevance when technical documents are concerned – but, rather perversely, for detective stories the opposite is often true. In such stories the author likes to play a game with readers, mentioning just in passing what they should (but usually do not) perceive to be a clue to solve the mystery. For example, in *Death in the Clouds*, the empty match box that will function as the decisive clue is only mentioned 4 times in the full text of the story: once quite casually in chapter 8, leaving the other allusions to chapter 26 where Poirot finally announces his conclusions. In contrast, terms referring to "blackmail", wrongly supposed to be the most likely motive of the crime, appear 15 times.

The other caveat is more serious. If the library is limited to a few stories, the automatic keyword-extraction services really are a convenient help to the provider. If a few words that deserve to be there are missing, they can without much effort be inserted by placing two windows side by side on the monitor's screen, one with the plot-summary, and the other with the domain Prolog module opened by the editing software (e.g. **Notepad++**), and then performing a manual cut-and-paste operation with the cursor.

But scaling-up by increasing the number of stories to their total of about 70, or to the 66 already available in **IMDB** plot-summaries[5], renders the automatic-extraction strategy unfeasible. We did an experiment, applying the **Yahoo** service to those 66 plot-summaries, and, after eliminating the duplicates, 879 distinct keywords resulted. Even if a careful provider should be able to drop many of them as irrelevant, too many would still remain to be displayed for choice in the rank-and-show process – inevitably discouraging the vast majority of users.

In the preliminary search through the Web to find the URLs of resources to be registered in the **KW-GPS** virtual library, a provider will ordinarily submit keywords that can be successfully employed by some popular Web-searching system. For various reasons it may be a good idea to include one extra keyword class, call it `init`, to preserve those keywords. On the one hand, the URL of a resource can be changed without redirection, or, even worse, the resource itself can be withdrawn. On the other hand, some users may diverge from the provider in matters of taste; for Poirot videos we are strongly biased in favour of David Suchet, but in this regard – and in general, since any system must remain open to customization – we feel obliged to leave room for other preferences.

For example, suppose the following `kws` clause keeps the keywords originally employed for locating resources for *Evil Under the Sun*:

```
kws(1, init, ['YouTube','Evil Under the Sun']).
```

then, if the line below[6] is executed, the system passes the list to **Google**, with an additional term in negated form, and a window is opened to display what is found in the Web:

```
?- kws(1,init,Ks), kw_google([-'Suchet'|Ks]).
```

wherein an admirer of Peter Ustinov will be glad to locate a version starred by this no less renowned artist.

To indicate, just one time, how to direct **SWI-Prolog** to access the Web, we reproduce here in full detail the code of the `kw_google` predicate:

---

[5] http://en.wikipedia.org/wiki/List_of_Agatha_Christie%27s_Poirot_episodes

[6] Improving end-user support is a continuing item in the agenda of our project, in order to avoid the need to express requests like this one in formal logic programming style.

```
kw_google(L) :-
  trg(L,X),
  tr_google(X).

tr_google(X) :-
  atom_concat('http://www.google.com/search?', X, Q),
  win_shell(open,Q).

trg([K],Kx) :- !,
  trg1(K,Kx).
trg([K|R],X) :-
  trg1(K,Kx),
  trg(R,X1),
  concat_atom([Kx,'&',X1],X).

trg1(K,Kx) :-
  (atomic(K), !, K1 = K;
   term_to_atom(K,K1)),
  (atom_concat('-',K2,K1), !,
   atom_concat('as_oq=&as_eq=',K2,Kx);
   atom_concat('as_q=',K1,Kx)).
```

3. Considerations on Keyword Repertoires

Keyword repertoires, either featuring ordinary words and phrases or more complex structures such as property:value pairs, may originate from strikingly different places. Among others, we can cite:

a) the original stories
b) plot summaries
c) terminology of the genre
d) user's personality traits

As noted in the previous section, the initial search for stories and respective resources must naturally use option (a), submitting to some popular Web searching system a list of terms expected to be in the originals. Searching instead over the shorter summaries has the advantage of efficiency, but there is a drawback: (b) is not always a faithful image of (a) because, for any given series of stories, the plot summaries are usually contributed by different people, with different idiosyncrasies.

Option (c) is particularky attractive, since with a limited number of terms the stories can be meaningfully characterized, and neatly compared with each other. For folktales, one may take the 31 functions described in [8] or (some subset of) the many types and motifs of the index compilation in [9]; for drama in general, 36 situations have been identified in [10].

Story segments (scenes), named after a dramatic situation (such as those enumerated in [10]) and with keyword classes indicating preconditions and postconditions, can be chained together to form branching plot sequences (comparable to business process *workflows*[7]), furnishing to authors a suitable storyboard scheme. For instance, situation `32` (`'mistaken jealousy'`) can lead to `26` (`'crimes of love'`), then branching out to either `16` (`madness`) or `34` (`remorse`).

Even to characterize the "story" of sportive games there exist official lists of events, sometimes called *scouts*. The *Fédération Internationale de Volleyball* (*FIVB*)[8] shows in its website the statistics of each game, which can easily be represented in property:value format, where the properties are scouts including `attack`, `block`, `serve`, and the values (for each player and the totals for the two contending teams) are the number of points gained through each of these skills.

For our Poirot examples we utilized, as seen, a number of properties, such as `motive`, `clue`, etc., commonly associated with the genre of detective stories. Surely several other properties in the same line could be added, taken both from studies on fictional works (e.g. the various phases of the crime and the investigation narratives, following Todorov's scheme, as in chapter 5, The Detective Film, of [11]) and on criminal law (e.g. `mitigating` and `aggravating` circumstances, as in [12]). When adding legal terms, however, one may find advisable to keep compatibility with the author's language, which is not always rigorously correct in this regard – for instance, in *Taken at the Flood*, Agatha Christie allows Poirot to discount as a mere accident what would still draw a verdict of involuntary manslaughter.

In line with option (d), the stereotype-based recommendation strategy reported in [13] offers a fascinating possibility, relating the personalities of users to the stories that they are supposed to like. A promising way to implement this notion is to assign to each story as property:value keywords the traits of the **Big Five** [14] proposal,

[7] http://link.springer.com/chapter/10.1007%2FBFb0020545#page-1
[8] http://www.fivb.org/

with percentile intervals as values. Short tests available in the Web[9] would measure the five percentiles of the current user, to be matched against the intervals estimated (roughly, to begin with and refined through usage, like Rich's **Grundy** system did) for the stories in the virtual library. Another intriguing possibility is to revert the direction: providers with a psychology background may wish to evaluate a person's **Big Five** percentiles by finding what stories the person has accessed, of which an additional **KW-GPS** module could keep record in files with appropriately detailed entries.

Returning to the possible advantages of property:value pairs over ordinary words and phrases, we should adduce that, in the realm of information systems, this format suits the conceptual apparatus of the *Entity-Relationship* model [15]. Repeating the same variable in two or more keywords is clearly an effective way, equivalent to the natural join of relational databases, to represent relationships. Furthermore the association of story/property/value recalls the subject/property/object structure of **RDF** triples [16].

A general remark is in order here: taking advantage of the ability to create multiple classes, providers can always accomodate keywords of different origins and different formats. Finally, domains outside the scope of digital entertainment are also amenable to keyword indexing. The subject index of a book is an obvious case, and the front page of technical papers must necessarily contain keywords that precisely characterize their subject – we shall add a brief remark on that at the end of section 5.

4. The rank-and- show Facility in Mobile Devices

In order to run the rank-and-show process in mobile devices (tablets and cell-phones) we developed an **Android** application that provides a graphical user interface to the **KW-GPS** system (Figure 2). The mobile application is based on a client-server architecture, where the server hosts the **KW-GPS** system and provides access to its functions, and the client contains the mobile user interface that allows users to search and enjoy web resources provided by the system (Figure 3).

Fig.2. Graphical user interface of the mobile application.

In the mobile interface, keywords are organized and displayed in classes (`victim`, `crime` and `investigation`) (Figure 2a). Each class contains three types of keywords: optional (green button), required (blue button), and excluded (red button). When a keyword class and type is designated by the user, the respective list of keywords is displayed for selection (Figure 2b). After the intended keywords are selected, the server performs the rank-and-show process, and the results are shown in the mobile interface (Figure 2c). Looking at the resulting sequence, the user is then free to indicate one of the stories, not necessarily the best ranked, causing the corresponding web resources to be displayed (Figure 2d). The interface is automatically adjusted to the domain specified in the **KW-GPS** system. All the required information about keywords is retrieved from the server that hosts the **KW-GPS** system. In this way,

[9] http://www.ocf.berkeley.edu/~johnlab/bfi.php

keyword classes and lists are automatically created and labelled according to the specifics of the current domain. The communication between the mobile application and the **KW-GPS** server is performed through a TCP/IP connection.

Fig. 3. Using the mobile interface

In order to assess the mobile interface, we have conducted a user evaluation with 9 participants, 8 male and 1 female, aged 16 to 17. Seven of them have some knowledge about detective stories, and three of them know the detective stories of Poirot. All of the participants frequently use **Google Web Search**.

We asked participants to use both our mobile application (S) and **Google Web Search** through the default **Android** web browser (G) to find a Poirot detective story to their taste. Our aim was therefore to compare the proposed mobile interface with the most commonly used method of web search in mobile devices. In order to reduce learning effects, half of the participants used S first, and the other half used G first. On average, each session of S lasted 4.05 minutes (σ=0.86), and each session of G lasted 9.22 minutes (σ=1.48).

After using each version, the participants filled in a questionnaire with 26 questions derived from the **USE** Questionnaire [17]. We evaluated the system usefulness, as well as user satisfaction and how easy was the use of the system. Each statement was followed by a seven-point **Likert scale** ranging from "strongly disagree" (-3) through "neutral" (0) to "strongly agree" (+3). After having interacted with both systems, the participants were interviewed about their experience.

Figure 4 summarizes the results of the questionnaires. Both **Google Web Search** and the mobile version of our system obtained similar system usefulness. On the other hand, the mobile **KW-GPS** system clearly improved user satisfaction and ease of use. In the course of the interviews, the participants declared that the mobile **KW-GPS** system was easy to use, gave them more accurate results, and allowed them to find an interesting story without risking to enter into unknown web pages. In contrast, some participants pointed out that the **Google Web Search** gave them more freedom.

During the user experiment, we also collected some statistical data about the time users spent to complete the task, number of searches, and number of clicks on wrong results. As shown in Figure 5, the proposed mobile **KW-GPS** system clearly reduced in more than half the time needed to complete the task, and reduced substantially the number of searches and clicks in wrong results.

Although the results of the user study reported here are not entirely conclusive, due to the small number of participants, the positive user feedback is a welcome stimulus for the continuation of our development efforts.

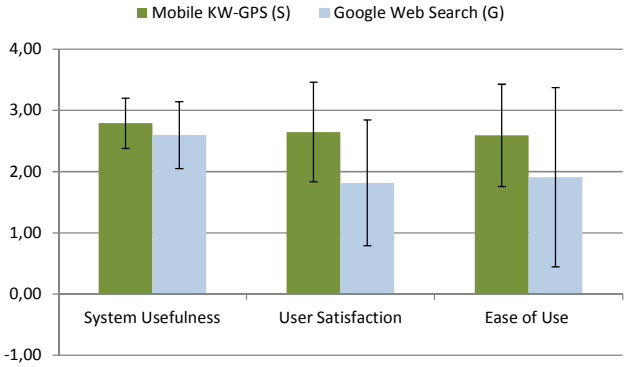

Fig. 4. Average number of points (within a 7-point Likert scale) of the system usefulness, user satisfaction, and ease of use, with error bars indicating standard deviation around the mean.

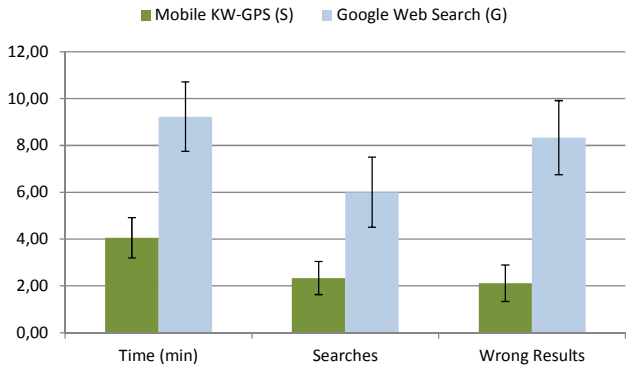

Fig. 5. Statistical data collected during the user evaluation about the time users spent to complete the task, number of searches, and number of clicks on wrong results.

5. Related Work

Keyword-based search is a well-studied problem in the area of information retrieval. Certain approaches [18, 19] have explored the use of keyword-based search for XML data. Formulating queries with keywords, those systems retrieve document fragments and use a ranking mechanism to increase the search result quality. Other approaches [20, 21] have investigated keyword-based search for Semantic Web and RDF data in order to provide ranked retrieval using content-based relevance estimation. Other works [22, 23] have extended simple keyword-based search with structured query capabilities.

Turning to the field of entertainment computing, one finds in [24] a sport video search and retrieval system, called DAVVI [25], with the capability of delivering sport video content for mobile and desktop devices. The system is based on automatic summarization and recommendation techniques, where sport videos are semi-automatically annotated with metadata extracted from live text commentary web pages, which include detailed textual information about the match minute-by-minute. Users can search and query for game events using keywords and phrases found in the live text commentaries. For example, in the soccer scenario a user can query for "*sliding tackles by Steven Gerrard*". The video annotations are then used by the search algorithm to create a playlist of videos containing the requested keywords. A similar system is described in [26].

There are several mobile applications that provide access to movie databases, such as the "*IMDb Movies & TV*"[10] [27], which is a mobile application developed for **Android**, **iOS** and **Windows Phone** that provides direct access to the **IMDB** movie information database, allowing users to search and navigate through a huge collection of movies and TV series. Another example is "*Movies by Flixster*" [28] for **Android**, **iOS** and **Windows Phone**, which allows users to browse and search for movies, read reviews and watch trailers in mobile devices. Several studies suggest the importance of domain specific search applications for mobiles devices. Both in [29] and in [30] it is emphasized that task-specific search applications are better to design in ways that more adequately serve the users' needs. In [31] it is argued that such applications allow users to retrieve documents with fewer interactions and less data traffic.

We ran an experiment to find out to what extent our **KW-GPS** work is in fact related to three among the projects surveyed above. Following the methods described in section 2, we put together a virtual library containing pointers to each referenced paper, the home page of the first author, and the home page of the project. Three keyword classes were considered containing, respectively, the keywords supplied by the authors, the official categories assigned by the publisher (ACM for the first two, and IEEE for the third), and some more indications which we labelled "complement" (tags in the ACM page, and INSPEC controlled and non controlled indexing terms in the IEEE page).

```
/* domain related references */

kw_classes([author, publisher, complement]).
thresholds(St, [A, P, C], T).
keyword_lists(St, [A, P, C]).

% LIBRARY

lib( 1, 'NAGA: harvesting, searching and ranking knowledge',
        [paper: 'http://dl.acm.org/citation.cfm?id=1376756',
         home: 'http://www.hpi.uni-potsdam.de/naumann/people/gjergji_kasneci.html',
         proj: 'http://suchanek.name/work/publications/index_e.php']).
```

[10] http://www.youtube.com/watch?v=IVMylQEJUGs

```
lib( 2, 'DAVVI: a prototype for the next generation multimedia entertainment platform',
        [paper: 'http://dl.acm.org/citation.cfm?id=1631482',
         home: 'http://site.uit.no/iad/',
         proj: 'http://www.youtube.com/watch?v=cPtvZ2kbt0w']).

lib( 3, 'vESP: A Video-Enabled Enterprise Search Platform',
        [paper: 'http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5635924&tag=1',
         home: 'http://www.iad-center.com/',
         proj: 'http://www.iad-center.com/publications/vesp-a-video-enabled-enterprise-search-
platform']).


% KEYWORDS

kws(1, author, ['semantic search', 'entities', 'relationships', 'ranking',
                'user  interface']).
kws(1, publisher, ['information retrieval']).
kws(1, complement, ['algorithms', 'design', 'experimentation',
                    'information search and retrieval']).

kws(2, author, ['video search and recommendation', 'annotation', 'personalization',
                'torrent-like dissemination']).
kws(2, publisher, ['video summarization', 'image and video acquisition',
                   'video segmentation']).
kws(2, complement, ['design', 'experimentation', 'performance', 'torrent-like',
                    'dissemination', 'video']).

kws(3, author, ['video enriched search result', 'user study']).
kws(3, publisher, ['computer architecture', 'indexes', 'multimedia communication',
                   'prototypes', 'search engines', 'streaming media', synchronization]).
kws(3, complement, ['multimedia computing', 'video retrieval',
                    'commercial enterprise search engine', 'disruptive multimedia service',
                    'powerpoint presentations', 'textual-oriented query results',
                    'user query', 'video-enabled enterprise search platform']).
```

Once this module was loaded, we executed the `rank` command, choosing the keywords that, in our opinion, were also applicable to our paper:

```
?- rank(S).

1:annotation
2:entities
3:personalization
4:ranking
5:relationships
6:semantic search
7:torrent-like dissemination
8:user interface
9:user study
10:video enriched search result
11:video search and recommendation
choose for author: 4, 9, 11

1:computer architecture
2:image and video acquisition
3:indexes
4:information retrieval
5:multimedia communication
6:prototypes
7:search engines
8:streaming media
9:synchronization
10:video segmentation
11:video summarization
choose for publisher: 4

1:algorithms
2:commercial enterprise search engine
3:design
4:disruptive multimedia service
5:dissemination
6:experimentation
7:information search and retrieval
8:multimedia computing
9:performance
10:powerpoint presentations
11:textual-oriented query results
12:torrent-like
13:user query
14:video
```

```
15:video retrieval
16:video-enabled enterprise search platform
choose for complement: 7
```

obtaining the following response from the system:

```
NAGA: harvesting, searching and ranking knowledge with 3 hits
vESP: A Video-Enabled Enterprise Search Platform with 1 hits
DAVVI: a prototype for the next generation multimedia entertainment platform with 1 hits

S = [1, 3, 2].
```

Regardless of its validity, the experiment suggests that our methods can also be tried on "serious" applications, in this case involving technical papers. And, since these are all digital entertainment papers, we can safely claim that, even here, we did not lose our focus on the topic at hand...

6. Concluding Remarks

Originally conceived as a tool to organize private virtual libraries of Poirot detective-stories, providing a multiple aspect keyword-based index mechanism to end users, the development of the **KW-GPS** system led us to a closer study of keywords as story descriptors.

In particular, it convinced us that some of our design decisions, especially the adoption of the property:value format, the inclusion of variable parameters to establish links between keywords, and the use of story templates to represent motifs, effectively open possibilities that seem profitably applicable to other domains. Not only casual users but even prospective authors of new stories may find that tools based on these notions are helpful to start creating new stories, by reusing and extending what the registered stories have to offer.

Regarding the modular architecture of the implemented system, it should be stressed that it aims at a two-sided purpose. Firstly, the simple `rank-and-show` process, running both in desktop/laptop computers and in mobile devices, was designed having in mind what we thought end-users would like to have available, and would find easy to handle in their leisure hours – and eventually would feel natural to share with other people. Secondly, the other basic commands, `similar` and `select`, as well as their extensions running on story templates, seemed adequate to be incorporated in larger applications, to be built by designers with programming expertise, with the help of providers with sound domain knowledge.

Particularly for applications employing artificial intelligence methods, the availability of a logic programming language was for us, and should continue to be for projects utilizing our proposal, a major asset. Among its unique features are the ability to formulate rules in logic and the excellent pattern-matching capability built into the Prolog interpreter. Moreover, the **SWI-Prolog** software, used to write our programs, features interfaces to other languages and environments. In other projects we used their **ODBC** interface to work on **Oracle** databases[11], and their interface library for **Java** to open and control windows for displaying images[12], whilst in the present project we resorted to their handy **HTTP** support[13] to access the Web.

Future research will explore other domains, and will submit the **KW-GPS** programs to more extensive user-evaluation experiments.

References

[1] A. Christie, Hercule Poirot - the Complete Short Stories. London: Harper, 2008.

[2] T. Todorov, The Poetics of Prose. Cornell: Cornell University Press, 1977.

[3] G. Navarro, "A Guided Tour to Approximate String Matching," ACM Computing Surveys, vol. 33 (1), pp. 31-88, 2001.

[4] G. Lakoff, M. Johnson, "Metaphors We Live", University of Chicago Press, 1980.

[5] K. Knight, "Unification: A Multidisciplinary Survey," ACM Computing Surveys, vol. 21(1), pp. 93-124, 1989.

[6] G. Fauconnier, M. Turner, The Way We Think: Conceptual Blending and the Mind's Hidden Complexities. Basic Books, 2002.

[7] H.C. Wu, R.W.P. Luk, K.F. Wong, K.L. Kwok, "Interpreting TF-IDF Term Weights as Making Relevance Decisions," ACM Transactions on Information Systems, vol. 26 (3), pp.1-13, 2008.

[8] V. Propp, Morphology of the Folktale. S. Laurence (trans.). University of Texas Pressm, 1968.

[9] A. Aarne, S. Thompson, The Types of the Folktale. Suomalainen Tiedeakatemia, 1987.

[10] G. Polti, The Thirty-Six Dramatic Situations. L. Ray (trans.). James Knapp Reeve, 1924

[11] D. Bordwell, Narration in the Fiction Film. Madison: University of Wisconsin Press, 1985.

[11] ftp://ftp.inf.puc-rio.br/pub/docs/techreports/11_11_furtado.pdf
[12] http://dl.acm.org/citation.cfm?doid=1658866.1658874
[13] http://www.swi-prolog.org/pldoc/package/http.html

[12] C.B. Hessick, Motive Role in Criminal Punishment. Southern California Law Review, pp. 89-150, 2008.

[13] E. Rich, “User modeling via stereotypes,” Cognitive Science, vol. 3, pp. 329-354, 1979.

[14] S.D. Gosling , P.J. Rentfrow, W.B. Swann, "A very brief measure of the Big-Five personality domains," Journal of Research in Personality, vol. 37, pp. 504–528, 2003.

[15] C. Batini, S. Ceri, S. Navathe, Conceptual Design - an Entity-Relationship Approach. Redwood: Benjamin Cummings, 1992.

[16] K.K. Breitman, M.A. Casanova, W. Truszkowski, Semantic Web - Concepts, Technologies and Applications. London: Springer, 2007.

[17] A. Lund, Measuring Usability with the USE Questionnaire. Usability and User Experience Newsletter of the STC Usability SIG, 2001, Available at: http://www.stcsig.org/usability/newsletter/0110_measurin g_with_use.html

[18] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, “XRANK: ranked keyword search over XML documents,” Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM Press, New York, pp. 16-27, 2003.

[19] Z. Liu, J. Walker, and Y. Chen, “XSeek: a semantic XML search engine using keywords,” Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 1330-1333, 2007.

[20] L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng, and P. Kolari, “Finding and ranking knowledge on the semantic web,” Proceedings of the 4th International Semantic Web Conference, pp. 156-170, 2005.

[21] G. Cheng, W. Ge, T. Qu, “Falcons: searching and browsing entities on the semantic web,” Proceeding of the 17th International Conference on World Wide Web, ACM Press, New York, 1101-1102, 2008.

[22] G. Kasneci, F.M. Suchanek, G. Ifrim, S. Elbassuoni, M. Ramanath, G. Weikum, “NAGA: harvesting, searching and ranking knowledge,” Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, Canada, pp. 1285-1288, 2008.

[23] F. Mandreoli, R. Martoglia, G. Villani, W. Penzo, “Flexible query answering on graph-modeled data,” Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, ACM, Saint Petersburg, Russia, pp. 216-227, 2009.

[24] D. Scott, C. Gurrin, D. Johansen, G. Johansen, “Searching and Recommending Sports Content on Mobile Devices,” Proceedings of the 16th International Multimedia Modeling Conference, Chongqing, China, pp. 779-781, 2010.

[25] D. Johansen, H., Johansen, T. Aarflot, J. Hurley, A. Kvalnes, C. Gurrin S. Sav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwodz, P. Halvorsen, “DAVVI: A Prototype for the Next Generation Multimedia Entertainment Platform,” Proceedings of the 17th ACM international conference on Multimedia, ACM Press, New York, pp. 989-990, 2009.

[26] P. Halvorsen, D. Johansen, B. Olstad, T. Kupka, S. Tennøe, “vESP: A Video-Enabled Enterprise Search Platform,”. Proceedings of the 4th International Conference on Network and System Security, Melbourne, Australia, pp. 534-541, 2010.

[27] IMDB, Movie Apps for iPhone, Android, iPad, WP7 & iPod. Available at: http://www.imdb.com/apps/ [Accessed 20 June 2013]

[28] Flixster, Flixster Mobile. Available at: http://www.flixster.com/mobile/a pps/ [Accessed 20 June 2013]

[29] M. Kamvar, M. Kellar, R. Patel, Y. Xu, “Computers and iphones and mobile phones, oh my!: a logs-based comparison of search users on different devices,” Proceedings of the 18th international conference on World Wide Web, Madrid, Spain, ACM, pp. 801-810, 2009.

[30] T. Sohn, K.A. Li, W.G. Griswold, J.D. Hollan, “A diary study of mobile information needs,” Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, Florence, Italy, ACM, pp. 433-442, 2008.

[31] E.W.D. Luca, A. Nürnberger, “Supporting information retrieval on mobile devices,” Proceedings of the 7th international conference on Human computer interaction with mobile devices & services, Salzburg, Austria, ACM, pp. 347-348, 2005.

Appendix A

```
/*  KW_GPS                          */
/*  KEYWORD GUIDE TO POIROT STORIES */
/*==================================================================*/
/*  RANK-AND-SHOW DIRECT FACILITY   */
/*==================================================================*/

/* preparatory commands */

:- set_prolog_flag(verbose,silent).
:- style_check([-singleton,-discontiguous]).
:- set_prolog_flag(toplevel_print_options,[max_depth(50)]).
:- op(900,fy,not).
:- op(500,fx,*).

:- dynamic lib/3, kws/3, kw_classes/1, thresholds/3, keyword_lists/2.
:- retractall(lib(_,_,_)).
:- retractall(kws(_,_,_)).
:- retractall(kw_classes(_)).
:- retractall(thresholds(_,_,_)).
:- retractall(keyword_lists(_,_)).

/* calls to defined domains */
:- include(my_poirot_1).



% RANK-AND-SHOW STORIES

rank(S) :-
  nl,
  ask_kw(Kws),
  xsetof(T:St-Tit,
   (Rs,Kws,Ns)^
     (lib(St,Tit,Rs),
     comp_st(St,Kws,Ns,T),
     keyword_lists(St,Kws),
     thresholds(St,Ns,T)),
   Sts1),
  reverse(Sts1,Sts),
  findall(St,member(_:St-_, Sts),S),
  (S = [], !,nl, write('no stories found'),nl,nl;
   nl,
   forall(member(T:St-Tit,Sts),(write(Tit),write(' with '),write(T),write(' hits'),nl)),
   nl, nl).


% COLLECT AND SORT THE KEYWORDS

kws_coll(C,X) :-
  kw_classes(Cs),
  member(C,Cs),
  xsetof(K,(I,Ks)^(kws(I,C,Ks), member(K,Ks)),X1),
  kws_n(X1,X).

kws_n(X,Xn) :-
  kws_n1(X,Xn,1).

kws_n1([],[],_) :- !.
kws_n1([K|R],[I:K|S],I) :-
  J is I + 1,
  kws_n1(R,S,J).
```

```
% ASK USER TO SELECT FROM THE KEYWORD LISTS

ask_kw(Kwps) :-
  findall(Kwp,
    (kws_coll(C,Kws),ask_kw1(C,Kws,Kwp)),
   Kwps).

ask_kw1(C,X,Kwp) :-
  (not ground(X), !,
   copy(X,Xc),
   varnames(Xc);
   Xc = X),
  forall(member(Ki,Xc),(write(Ki), nl)),
  (kw_classes(Cs),
   member(C,Cs),!,
   atom_concat('choose for ',C,Ch);
   Ch = 'choose'),
  length(X,N),
  write(Ch), write(': '),read_sel(P,N),
  (P == [],!,Kwp = [];
   pick_k(P,X,Kwp)),
   nl.

read_sel(P,N) :-
  read_line1(P1),
  (P1 == nil,!,P = [];
   name(P1,Nx),
   (Nx = [91|_], !, Ny = Nx;
    Nx = [32,91|_], !, Ny = Nx;
    Ny = [91|Nx]),
   reverse(Ny,Nr),
   (Nr = [93|_], !, N1 = Ny;
    append(Ny,[93],N1)),
   replace(43,42,N1,N2),
   name(P2,N2),
   term_to_atom(P3,P2),
   (P3 = [_|_],
    memberx(Pi,P3),
    not between(1,N,Pi),!,
    write('please, only numbers between 1 and '),write(N),write(' - choose:'),
    read_sel(P,N);
    P = P3)).

memberx(T,L) :-
  member(Tx,L),
  (integer(Tx),
   T is abs(Tx);
   not integer(Tx),
   Tx =.. [*,T]).

pick_k(P,Kw,Kwp) :-
  xsetof(K,(I,Ip,K1)^(member(Ip,P),
    (Ip = *I,member(I:K1,Kw),K = *K1;
    not (Ip = *I),Ip < 0,I is abs(Ip),member(I:K1,Kw),K = -K1;
    not (Ip = *I),Ip > 0,member(Ip:K,Kw))),
    Kwp).


% MATCH CHOSEN KEYWORDS TO THOSE OF THE STORIES

comp_st(St,Kws,Ns,T) :-
  findall(Kl,kws(St,_,Kl),Ks),
  comp_st1(St,Kws,Ks,Ns,T).

comp_st1(St,[],[],[],0) :- !.
comp_st1(St,[[]|R1],[Kst|R2],[0|R3],T) :- !,
  comp_st1(St,R1,R2,R3,T).
comp_st1(St,[Kwp|R1],[Kst|R2],[N|R3],T) :-
  comp_st2(Kwp,Kst,N),
  comp_st1(St,R1,R2,R3,T1),
  T is T1 + N.
```

```
comp_st2(Kwp,Kst,N) :-
  (member(*K,Kwp),not member(K,Kst),!,fail;
   member(-K,Kwp),member(K,Kst),!,fail;
   xsetof(K,(member(*K,Kwp);member(K,Kwp), not (K = *_)),Kwp1),
   sort(Kst,Kst1), count_hit(Kwp1,Kst1,N)).

count_hit([],Kst,0) :- !.
count_hit(Kwp,[],0) :- !.
count_hit([K|R],[K|S],N) :- !,
  count_hit(R,S,N1),
  N is N1 + 1.
count_hit([K1|R],[K2|S],N) :-
  K1 @> K2,!,
  count_hit([K1|R],S,N).
count_hit([K1|R],[K2|S],N) :-
  count_hit(R,[K2|S],N).


% EXPLORING ASSOCIATED PAGES

title(St,Tit) :-
  lib(St,Tit,_).

show(T,St) :-
  lib(St,_,Rl),
  member(T:U,Rl),
  tr_web(U).

has_resources(St,Tit,Rs) :-
  xbagof(R,(Rl,U)^(lib(St,Tit,Rl),member(R:U,Rl)),Rs).


% UTILITIES

tr_web(Url) :-
  atom_concat('"c:\\program files\\internet explorer\\iexplore.exe" ',Url,Arg),
  shell(Arg).

show(wn,W) :-
  atom_concat('http://wordnetweb.princeton.edu/perl/webwn?s=',W,U),
  tr_web(U).

show(db,W) :-
  name(W,L),
  replace(32,95,L,L1),
  name(W1,L1),
  atom_concat('http://dbpedia.org/page/',W1,U),
  tr_web(U).

xsetof(X,Y,Z) :-
  (setof(X,Y,Z), !; Z = []).

xbagof(X,Y,Z) :-
  (bagof(X,Y,Z), !; Z = []).

replace(_,_,[],[]) :- !.
replace(X,Y,X,Y) :-
  atomic(X), !.
replace(X,Y,X,Y) :-
  var(X), !.
replace(X,Y,Z,Z) :-
  atomic(Z), !,
  not (X = Z), !.
replace(X,Y,[Z|L],[Z1|L1]) :- !,
  replace(X,Y,Z,Z1),
  replace(X,Y,L,L1).
replace(X,Y,Z1,Z2) :-
  Z1 =.. [F|L],
  replace(X,Y,F,F1),
  replace(X,Y,L,L1),
```

```
  Z2 =.. [F1|L1].

read_line(A) :-
  rl([],L),
  (L == [], A = nil;
   not (L == []),
   name(A,L)).

read_line1(A) :-
  rl([32],L),
  (L == [32], A = nil;
   not (L == [32]),
   name(A,L)).

rl(L,L1) :-
  get0(A),
  (not (A == 10), !,
   append(L,[A],L2),
   rl(L2,L1);
   L1 = L).

v_on(X,[Y|Z]) :-
  var(Y),!,
  (X == Y ;
   v_on(X,Z)).
v_on(X,[X|_]).
v_on(X,[Y|Z]) :-
  v_on(X,Z).

no_dup([X|Xs],Ys) :-
  v_on(X,Xs), !,
  no_dup(Xs,Ys).
no_dup([X|Xs],[X|Ys]) :-
  not v_on(X,Xs),
  no_dup(Xs,Ys).
no_dup([],[]).

v_flat(A,B,[A|B]) :-
  var(A), !.
v_flat(A,B,B) :-
  atomic(A), !.
v_flat([A|R],B,C) :- !,
  v_flat(R,B,T),
  v_flat(A,T,C).
v_flat(A,B,C) :-
  A =.. [F|R], !,
  v_flat(R,B,C).

listvar(X,Y) :-
  v_flat(X,[],T1),
  reverse(T1,T2),
  no_dup(T2,T3),
  reverse(T3,Y).

varnames(A) :-
  listvar(A, B),
  length(B, C),
  D is 64+C,
  findall(E, (between(65, D, F), name(E, [F])), G),
  B=G.

copy(X,Y) :-
  listvar(X,L1),
  cop1(L1,L),
  cop2(X,L,Y), !.

cop1([],[]).
cop1([V|R],[[V,W]|S]) :-
  cop1(R,S).

cop2(X,L,Y) :-
```

```
  (atomic(X) -> Y = X;
   var(X) -> (cop4(X,X1,L), Y = X1);
   functor(X,F,N) -> (functor(Y,F,N), cop3(0,N,X,L,Y))).

cop3(N,N,X,L,Y) :- !.
cop3(I,N,X,L,Y) :-
  J is I + 1,
  arg(J,X,A),
  arg(J,Y,B),
  cop2(A,L,B),
  cop3(J,N,X,L,Y).

cop4(X,X1,[[Y,X1]|R]) :-
  X == Y, !.
cop4(X,X1,[P|R]) :-
  cop4(X,X1,R).


/*====================================================================*/
/*  session-administration commands                                   */
/*====================================================================*/

session :-
  user_int,
  nl,write('do you want to try another selection? (y/n): '),read_line(R),
  (R = n, !, halt;
   R = y, session).

ask_show(S) :-
   (S = [],!;
    findall(Tit,(member(Is,S),lib(Is,Tit,_)),Stc1),
   append(Stc1,[end],Stc),
    length(Stc,L1),
    kws_n(Stc,Stcn),nl,
   forall(member(Sti,Stcn),(write(Sti), nl)),
    write('choose a story: '),read_line(N),
    (N = L1, !;
    N = end, !;
     member(N:Tit,Stcn),
    lib(St,Tit,Rs),
     findall(R,member(R:_,Rs),Rc1),
    append(Rc1,[end],Rc),
    length(Rc,L2),
     kws_n(Rc,Rcn),
    ask_show1(St,Rcn,L2),
    ask_show(S))).

ask_show1(St,Rcn,L2) :-
  nl,
  forall(member(Ri,Rcn),(write(Ri), nl)),
  write('choose what you want to see: '),read_line(I),
  (I = L2, !;
   I = end, !;
   member(I:R,Rcn),
   show(R,St),
   ask_show1(St,Rcn,L2)).

user_int :-
   rank(S),
   write('************************************************************'),nl,
   write('This is what was found. Please select from the options below'), nl,
   write('************************************************************'),nl,
   ask_show(S).

:- session.
```

Appendix B

```
/* domain My Poirot 1 */

kw_classes([victim, crime, investigation]).
thresholds(St, [V, C, I], T).
keyword_lists(St, [V, C, I]).


% LIBRARY

lib( 1, 'Evil Under the Sun',
        [plot_summary:  'http://www.imdb.com/title/tt0676148/plotsummary?ref_=tt_ql_5',
         wiki: 'http://en.wikipedia.org/wiki/Evil_Under_the_Sun#Plot',
         video: 'http://www.youtube.com/watch?v=siHNgQPaEnc']).

lib( 2, 'Cards on the Table',
        [plot_summary: 'http://www.imdb.com/title/tt0676143/plotsummary?ref_=tt_ql_5',
         wiki: 'http://en.wikipedia.org/wiki/Cards_on_the_Table#Plot_summary',
         video: 'http://www.youtube.com/watch?v=BH_IVwIiUEw']).

lib( 3, 'The Mysterious Affair at Styles',
        [plot_summary: 'http://www.imdb.com/title/tt0100216/plotsummary?ref_=tt_ql_5',
      wiki: 'http://en.wikipedia.org/wiki/The_Mysterious_Affair_at_Styles#Plot_summary',
         video: 'http://www.youtube.com/watch?v=2K2C_EXuWao',
           full_text: 'http://www.gutenberg.org/files/863/863-h/863-h.htm']).

lib(4, 'The Chocolate Box',
       [plot_summary: 'http://www.imdb.com/title/tt0676169/plotsummary?ref_=tt_ql_5',
       wiki: 'http://en.wikipedia.org/wiki/Poirot''s_Early_Cases#The_Chocolate_Box',
       video: 'http://www.youtube.com/watch?v=Dk0lFG4k6vA']).


% KEYWORDS

kws(1, victim, [gender: female, marital_status: married, occupation: actress,
                character: credulous, swindled_by: 'younger man', age_bracket: 30,
                  economic_status: rich]).
kws(1, crime, [action: murder, means: strangulation, place: beach, motive: 'financial gain',
               circumstance: 'holiday season', companion: (lover,'younger man')]).
kws(1, investigation, [clue: 'character of the victim', snag: 'time of death',
                       tactic: 'break self-control']).


kws(2, victim, [gender: male, marital_status: single, occupation: 'art collector',
                character: bizarre, age_bracket: 40, economic_status: rich]).
kws(2, crime, [action: murder, means: stabbing, place: 'drawing room',
               motive: 'avoid accusation', circumstance: 'dinner party']).
kws(2, investigation, [clue: 'bridge scores', tactic: 'deceiving trick']).


kws(3, victim, [gender: female, marital_status: married, age_bracket: 70,
                character: credulous, economic_status: 'large fortune',
                attitude: autocratic, swindled_by: 'younger man',
                occupation: 'social work']).
kws(3, crime, [action: murder, means: poisoning, place: bedroom, motive: 'financial gain',
               circumstance: 'medical treatment', companion: (someone,'younger man')]).
kws(3, investigation, [clue: 'incriminating letter', tactic: 'expose evidence',
                       snag: 'time of death']).


kws(4, victim, [gender: male, age_bracket: 30, character: evil, occupation: politician,
                economic_status: middle]).
kws(4, crime, [action: execution, means: poisoning, motive: 'moral reasons', place: study,
               circumstance: conversation]).
kws(4, investigation, [clue: chocolates, snag: 'mistaken suspect', tactic: confession]).
```