The AMOS Project

IST-2001-34717

The Interface Internals

Deliverable D14

Responsible person: José F. Morales, Edison F. Mera, Manuel Carro, and Jesús Correas, Technical University of Madrid ({jfran,edison,jcorreas}@clip.dia.fi.upm.es, mcarro@fi.upm.es)

Date of current release:	December 2003
Type of deliverable:	Report

Abstract

This document describes the internal interface that is needed within the Amos tool to allow communication between the matching engine and the web-based user interface. In order to make the search web form more user-friendly, some searching parameters needed by the matching engine have been abstracted using common linguistic expressions that reflect in an informal way search requirements. In [GCM04] and [CGC+04] the search user interface is explained in depth. This document describes the code and interfaces involved in such communication.

Contents

1 Introduction

The matching engine [CMM04] uses a number of parameters in order to guide the searching algorithm. These parameters are represented by means of an internal query language interpreted by the matching engine. The part of the external interface related to the search of assemblies translates the user query to that language, and invokes the matching engine to obtain the assemblies that satisfy the searching parameters specified by the user.

In the Amos system there is a clear conceptual distinction between the user interface and the algorithms and techniques involved in the search for package assemblies. In order to provide a seamless integration between both of the main components of the system, the communication protocol between the web-based user interface and the Amos matching engine is performed by means of Ciao Prolog module interfaces. This approach improves the functionality of the system while at the same time eases the implementation.

The searching user interface is based on a Ciao Prolog CGI program that generates the HTML forms for interacting with the user, transforms the user input into the internal query language, and performs the query on the Amos database, presenting the resulting assemblies to the user with a dynamically generated html page.

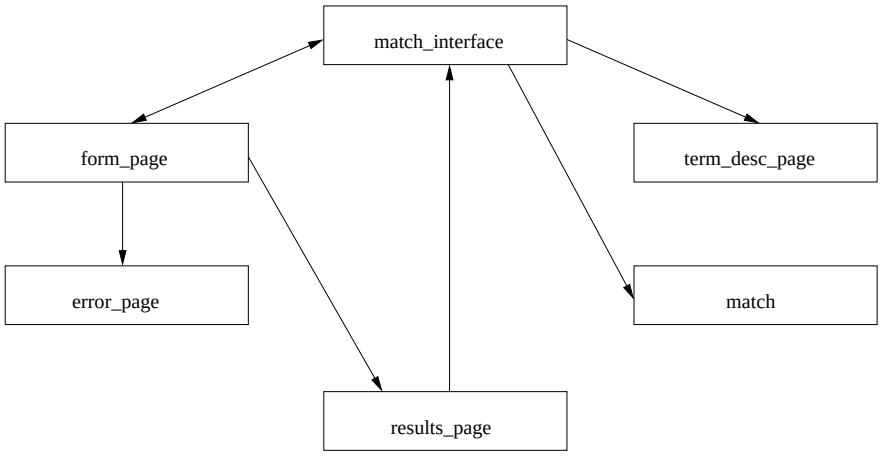

Figure 1: Module description of the search interface internals

The matching engine and the user interface for searching have been implemented through a set of modules, represented in Figure 1. Each module provides a different function, as follows:

- The module `match_interface` is the starting point of this part of the Amos

system. It is a CGI program that generates the HTML pages for requesting query parameters and displaying the results of a search performed by the matching engine, transforms the query input by the user and sends it to the matching engine. This module uses the rest of the modules, as shown in 1. This module uses `form_page` for actual HTML page generation.

- `form_page` generates the main search page. The main search page is composed by a form to enter search parameters and options, and can be followed by a section with the resulting assemblies of a previous page, or a section with informative messages about problems found in the previous user input. In order to do this in a modular way, it uses the modules `error_page` and `results_page`. `form_page` also accesses the Amos database to load into the search page the dictionary terms to facilitate user interaction.

- The modules `error_page` and `results_page` generate dynamically the informative messages and the list of assemblies that results from a given search, respectively.

- Finally, `match` contains the code of the matching engine. It receives a query in the internal query language, and produces a list of assemblies which satisfy the given query.

In following sections these modules are described in depth.

2 match_interface (library)

This module performs the interaction between the user and the matching engine by means of a web interface based on efficient CGI scripts. When the web interface starts for the first time, an initial query with default search options is shown. The user can then select the description terms to find, and adjust the searching parameters as well, in order to get more precise results. When the user eventually performs the query (by pressing the appropiate button on the web page), assemblies satisfying the query parameters are obtained, and a result page is shown to the user. The assemblies that result from a query evaluation are presented to the user together with another search form, in order to enable incremental searches. The information introduced by the user is kept in this new form to let him refine the search.

The underlying implementation of the search web interface is based on a single HTML form which encodes the application state given by the `values/1` procedure. Basically, this procedure abstracts away the implementation details of the HTML page, mapping each state element as an HTML input, which can be either hidden, visible or a human readable menu which disguises the representation of the matching engine parameters. Depending on whether the application has been started or not, the main procedure `form_handling/0` executes one of these actions:

1- to emit an initial form with default values, or

2- to receive the CGI input, extract the event identifier (represented by an element of type `id`), input fields and previous state encoded as input fields, process them into a new state, and generate an updated form.

The actions of the web interface which require the Amos matching engine are `Search`, `Next page` (to show the next page of search results when the last query resulting assemblies do not fit a single page) or `Previous page` (to show the previous page of search results). The connection with the matching engine to retrieve the search solutions is made by `make_search/2`. Any other action simply updates the input state without performing any search.

The search parameters are translated to the internal query language needed by the matching engine. The translation is straightforward, with the exception of the parameters to control the search heuristics (described in `sort_method/1`). This interface makes use of linguistic modifiers to link the meaning of numeric weights (explained in `match`) in order to make them more intuitive for humans, as follows:

- number of packages: very few (-2), few (-1), any number or (0), a few (1), many (2)
- unsatisfied requirements: very few (-2), few (-1), any number or (0)
- auxiliary requirements: very few (-2), few (-1), any number or (0)
- number of capabilities: very few (-2), few (-1), any number or (0), a few (1), many (2)
- ratio of fullfilled capabilities: a very small (-2), a small (-1), any (0), a large (1), a very large (2)

2.1 Usage and interface (`match_interface`)

- **Library usage:**

  ```
  :- use_module(library(match_interface)).
  ```

- **Exports:**

  - *Predicates:*

    `start/0`.

  - *Regular Types:*

    `id/1`, `values/1`, `results/1`.

- **Other modules used:**

  - *Application modules:*

    `search(form_page)`, `search(term_desc_page)`, `search(match)`, `search(utils)`, `amos(configuration)`.

  - *System library modules:*

    `pillow/http`, `pillow/html`, `prolog_sys`, `lists`, `aggregates`, `terms`, `pillow/pillow_types`.

  - *Internal (engine) modules:*

    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

2.2 Documentation on exports (`match_interface`)

start/0: PREDICATE

Usage:

- *Description:* Main procedure to start the CGI.

id/1: REGTYPE

Usage: `id(X)`

- *Description:* Identifier of the action.

values/1: REGTYPE

```
values(values(Terms,Ini,Len,SortMethod,Mode,InP,ExP)) :-
        atm(Terms),
        num(Ini),
        num(Len),
        sort_method(SortMethod),
        match_mode(Mode),
        atm(InP),
        atm(ExP).
```

Usage: `values(X)`

- *Description:* Internal state of the search CGI.

results/1: REGTYPE

`results(Results)`

A list of assemblies which are the result of a search, plus the total number of solutions `Total` and the computation time `Time`.

2.3 Documentation on internals (`match_interface`)

form_handling/0: PREDICATE

Usage:

- *Description:* Gets the CGI input and sends back the corresponding HTML page to the client web browser.

process_form/1: PREDICATE

Usage: `process_form(Info)`

- *Description:* Generates and sends an HTML page.

– *Calls should, and exit will be compatible with:*

`Info` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (`form_dict/1`)

– *The following properties should hold at call time:*

`Info` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

`Info` is currently ground (it contains no variables). (`ground/1`)

decide_page/3: PREDICATE

Usage: `decide_page(Id, Info, HtmlCode)`

– *Description:* Processes form input information `Info` and the action identifier `Id`, and generates `HtmlCode`.

– *Calls should, and exit will be compatible with:*

Identifier of the action. (`id/1`)

`Info` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (`form_dict/1`)

`HtmlCode` is a term representing HTML code. (`html_term/1`)

– *The following properties should hold at call time:*

`Id` is currently ground (it contains no variables). (`ground/1`)

`Info` is currently ground (it contains no variables). (`ground/1`)

`HtmlCode` is a free variable. (`var/1`)

– *The following properties hold upon exit:*

`Id` is currently ground (it contains no variables). (`ground/1`)

`Info` is currently ground (it contains no variables). (`ground/1`)

`HtmlCode` is currently ground (it contains no variables). (`ground/1`)

concat_with_spaces/2: PREDICATE

Usage: `concat_with_spaces(Xs, Y)`

– *Description:* `Y` is the concatenation of the elements of `Xs` separated with blank spaces.

– *Calls should, and exit will be compatible with:*

  `Xs` is a list of `atm`s. (`list/2`)

  `Y` is an atom. (`atm/1`)

– *The following properties should hold at call time:*

  `Xs` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

  `Xs` is currently ground (it contains no variables). (`ground/1`)

  `Y` is currently ground (it contains no variables). (`ground/1`)

default_values/1: PREDICATE

Usage: `default_values(Values)`

– *Description:* `Values` is the default state of a newly loaded search page.

– *Calls should, and exit will be compatible with:*

  Internal state of the search CGI. (`values/1`)

– *The following properties hold upon exit:*

  `Values` is currently ground (it contains no variables). (`ground/1`)

get_values/2: PREDICATE

Usage: `get_values(Info, Values)`

– *Description:* Decodes the information `Info` obtained by the CGI from the input form and returns it in `Values`.

– *Calls should, and exit will be compatible with:*

  `Info` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (`form_dict/1`)

  Internal state of the search CGI. (`values/1`)

– *The following properties should hold at call time:*

  `Info` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

  `Info` is currently ground (it contains no variables). (`ground/1`)

  `Values` is currently ground (it contains no variables). (`ground/1`)

no_results/1: PREDICATE

Usage: `no_results(Results)`

- *Description:* `Results` is an empty result
- *Calls should, and exit will be compatible with:*
  `results(Results)` (`results/1`)
- *The following properties hold upon exit:*
  `Results` is currently ground (it contains no variables). (`ground/1`)

make_search/2: PREDICATE

Usage: `make_search(Values, Results)`

- *Description:* Converts the search parameters `Values` stored in the CGI state to the internal query language and performs the query.
- *Calls should, and exit will be compatible with:*
  Internal state of the search CGI. (`values/1`)
  `results(Results)` (`results/1`)
- *The following properties should hold at call time:*
  `Values` is currently ground (it contains no variables). (`ground/1`)
- *The following properties hold upon exit:*
  `Values` is currently ground (it contains no variables). (`ground/1`)
  `Results` is currently ground (it contains no variables). (`ground/1`)

decode_package_list/2: PREDICATE

Usage: `decode_package_list(PA, P)`

- *Description:* `P` is the result of decoding `PA`.
- *Calls should, and exit will be compatible with:*
  `PA` is an atom. (`atm/1`)
  `P` is a list of `package`s. (`list/2`)
- *The following properties should hold at call time:*
  `PA` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

`PA` is currently ground (it contains no variables). (`ground/1`)

`P` is currently ground (it contains no variables). (`ground/1`)

check_request/2: PREDICATE

Usage: `check_request(Values, Problems)`

– *Description:* Checks the types of `Values` and generate a the possible list of problems `Problems`

– *Calls should, and exit will be compatible with:*

Internal state of the search CGI. (`values/1`)

`Problems` is a list of `problem`s. (`list/2`)

– *The following properties should hold at call time:*

`Values` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

`Values` is currently ground (it contains no variables). (`ground/1`)

`Problems` is currently ground (it contains no variables). (`ground/1`)

check_request__number/4: PREDICATE

Usage: `check_request__number(Name, Value, P0, P)`

– *Description:* Inserts the corresponding problem (identifing the culprit input `Name`) into the difference list `P0-P` if the type of `Value` is not a number.

– *Calls should, and exit will be compatible with:*

`Name` is an atom. (`atm/1`)

`Value` is an atom. (`atm/1`)

`P0` is a list of `problem`s. (`list/2`)

`P` is a list of `problem`s. (`list/2`)

– *The following properties should hold at call time:*

`Name` is currently ground (it contains no variables). (`ground/1`)

`Value` is currently ground (it contains no variables). (`ground/1`)

`P` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

`Name` is currently ground (it contains no variables). (`ground/1`)

`Value` is currently ground (it contains no variables). (`ground/1`)

`P0` is currently ground (it contains no variables). (`ground/1`)

`P` is currently ground (it contains no variables). (`ground/1`)

check_request__empty/4: PREDICATE

Usage: `check_request_empty(Name, Value, P0, P)`

– *Description:* Insert the corresponding problem (identifing the culprit input `Name`) into the difference list `P0-P` if the type of `Value` is empty.

– *Calls should, and exit will be compatible with:*

`Name` is an atom. (`atm/1`)

`Value` is an atom. (`atm/1`)

`P0` is a list of `problem`s. (`list/2`)

`P` is a list of `problem`s. (`list/2`)

– *The following properties should hold at call time:*

`Name` is currently ground (it contains no variables). (`ground/1`)

`Value` is currently ground (it contains no variables). (`ground/1`)

`P` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

`Name` is currently ground (it contains no variables). (`ground/1`)

`Value` is currently ground (it contains no variables). (`ground/1`)

`P0` is currently ground (it contains no variables). (`ground/1`)

`P` is currently ground (it contains no variables). (`ground/1`)

which_page_id/2: PREDICATE

Usage: `which_page_id(Info, Id)`

– *Description:* Obtains the identifier of the current page `Id` from `Info`. This identifier can be blank if the form is in its initial state or a particular value telling which button or action was selected by the user.

– *Calls should, and exit will be compatible with:*

  `Info` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (`form_dict/1`)

  Identifier of the action. (`id/1`)

– *The following properties should hold at call time:*

  `Info` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

  `Info` is currently ground (it contains no variables). (`ground/1`)

  `Id` is currently ground (it contains no variables). (`ground/1`)

3 form_page (library)

This module generates the initial search form, used to perform queries. The initial search form must be generated dynamically, as it depends on the description terms stored in the Amos database.

3.1 Usage and interface (`form_page`)

- **Library usage:**

  ```
  :- use_module(library(form_page)).
  ```

- **Exports:**

  - *Predicates:*

    `generate_form_page/4`,
    `generate_selected_termnames/1`.

- **Other modules used:**

  - *Application modules:*

    `portalfiles(web_utils)`,
    `portalfiles(standard_page)`, `search(utils)`,
    `search(results_page)`, `search(error_page)`,
    `search(match_interface)`, `search(balboa)`,
    `search(db_names)`, `balboa_term_desc`.

  - *System library modules:*

    `pillow/http`, `pillow/html`, `aggregates`, `lists`,
    `between`, `pillow/pillow_types`.

  - *Internal (engine) modules:*

    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`,
    `basic_props`, `basiccontrol`, `data_facts`, `exceptions`,
    `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`,
    `system_info`, `term_basic`, `term_compare`, `term_typing`.

3.2 Documentation on exports (`form_page`)

generate_form_page/4: PREDICATE

Usage: `generate_form_page(Values, PageId, Results, HTML)`

- *Description:* Generates in `HTML` the code of the main search page, using `Values` as the values to fill in the fields of the search form. This HTML page may contain results of previous queries, that can be specified in `Results`, if it is available.
- *Calls should, and exit will be compatible with:*

  Internal state of the search CGI. (`values/1`)

  Identifier of the action. (`id/1`)

  `results(Results)` (`results/1`)

  `HTML` is a term representing HTML code. (`html_term/1`)
- *The following properties should hold at call time:*

  `Values` is currently ground (it contains no variables). (`ground/1`)

  `PageId` is currently ground (it contains no variables). (`ground/1`)

  `Results` is currently ground (it contains no variables). (`ground/1`)
- *The following properties hold upon exit:*

  `Values` is currently ground (it contains no variables). (`ground/1`)

  `PageId` is currently ground (it contains no variables). (`ground/1`)

  `Results` is currently ground (it contains no variables). (`ground/1`)

  `HTML` is currently ground (it contains no variables). (`ground/1`)

generate_selected_termnames/1: PREDICATE

Usage: `generate_selected_termnames(TermNames2Add)`

- *Description:* Obtains the names of the selection lists of description terms in the search form page.
- *Call and exit should be* compatible *with:*

  `TermNames2Add` is a list of `atm`s. (`list/2`)
- *The following properties should hold upon exit:*

  `TermNames2Add` is currently ground (it contains no variables). (`ground/1`)

4 results_page (library)

This module generates a HTML page that contains resulting assemblies of a given query. This HTML page is embedded in the general search form page (generated by form_page), in order to allow sequences of queries.

4.1 Usage and interface (`results_page`)

- **Library usage:**

  ```
  :- use_module(library(results_page)).
  ```

- **Exports:**

  - *Predicates:*

    `results_page/3`.

- **Other modules used:**

  - *Application modules:*

    `amos(configuration)`, `search(utils)`, `portalfiles(standard_page)`, `portalfiles(web_utils)`, `search(match_interface)`, `search(balboa)`, `search(db_names)`, `balboa_term_desc`.

  - *System library modules:*

    `pillow/http`, `pillow/html`, `lists`, `terms`, `pillow/pillow_types`.

  - *Internal (engine) modules:*

    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

4.2 Documentation on exports (`results_page`)

results_page/3: PREDICATE

Usage: `results_page(Values, Results, HTML)`

- *Description:* Translates the search results stored in `Results` plus the search parameters `Values` given by the user in the search form to a formatted HTML view `HTML`
- *Calls should, and exit will be compatible with:*

  Internal state of the search CGI. (`values/1`)

  `results(Results)` (`results/1`)

  `HTML` is a term representing HTML code. (`html_term/1`)
- *The following properties should hold at call time:*

  `Values` is currently ground (it contains no variables). (`ground/1`)

  `Results` is currently ground (it contains no variables). (`ground/1`)
- *The following properties hold upon exit:*

  `Values` is currently ground (it contains no variables). (`ground/1`)

  `Results` is currently ground (it contains no variables). (`ground/1`)

  `HTML` is currently ground (it contains no variables). (`ground/1`)

5 error_page (library)

This module generates an HTML page showing a list of error messages.

5.1 Usage and interface (`error_page`)

- **Library usage:**

  ```
  :- use_module(library(error_page)).
  ```

- **Exports:**

  - *Predicates:*

    `error_page/2`.

  - *Regular Types:*

    `problem/1`.

- **Other modules used:**

  - *System library modules:*

    `pillow/http`, `pillow/html`, `hiordlib`, `pillow/pillow_types`.

  - *Internal (engine) modules:*

    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

5.2 Documentation on exports (`error_page`)

error_page/2: PREDICATE

Usage: `error_page(Problems, HTML)`

- *Description:* Generates an HTML page `HTML` for showing a list of problems `Problems` detected in the data introduced by the user in the search form page. It is used throughout the system to generate user friendly error messages when necessary.

– *Calls should, and exit will be compatible with:*

`Problems` is a list of `problem`s. (`list/2`)

`HTML` is a term representing HTML code. (`html_term/1`)

– *The following properties should hold at call time:*

`Problems` is currently ground (it contains no variables). (`ground/1`)

– *The following properties hold upon exit:*

`Problems` is currently ground (it contains no variables). (`ground/1`)

`HTML` is currently ground (it contains no variables). (`ground/1`)

problem/1: REGTYPE

```
problem(empty(X)) :-
        atm(X).
problem(no_num(X)) :-
        atm(X).
```

Usage:

– *Description:* Type for use in `error_page/2`.

6 term_desc_page (library)

This module generates a HTML page showing the description of all dictionary terms included in the Amos database.

6.1 Usage and interface (`term_desc_page`)

- **Library usage:**

  ```
  :- use_module(library(term_desc_page)).
  ```

- **Exports:**

  - *Predicates:*

    `generate_term_description/1`.

- **Other modules used:**

  - *Application modules:*

    `search(utils)`, `portalfiles(standard_page)`, `portalfiles(web_utils)`, `search(balboa)`, `search(db_names)`, `balboa_term_desc`.

  - *System library modules:*

    `pillow/http`, `pillow/html`, `aggregates`, `pillow/pillow_types`.

  - *Internal (engine) modules:*

    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

6.2 Documentation on exports (`term_desc_page`)

generate_term_description/1: PREDICATE

Usage: `generate_term_description(HTML)`

– *Description:* Generates a HTML page with the description of all dictionary terms included in the Amos database.

– *Call and exit should be* compatible *with:*

  `HTML` is a term representing HTML code. (`html_term/1`)

– *The following properties should hold upon exit:*

  `HTML` is currently ground (it contains no variables). (`ground/1`)

7 match (library)

This module contains the implementation of the matching engine. Given a query expressed in the Amos internal language, it produces a list of packages that satisfy the query.

7.1 Usage and interface (`match`)

- **Library usage:**

  ```
  :- use_module(library(match)).
  ```

- **Exports:**

  - *Predicates:*

    `match_sort_interval_list/9`.

  - *Regular Types:*

    `match_mode/1`, `match_sort_result/1`, `sort_method/1`, `weight_package_number/1`, `weight_unsatisfied_deps/1`, `weight_tighter_assemblies/1`, `weight_capabilities/1`, `weight_best_capabilities_ratio/1`, `package/1`, `capability/1`.

- **Other modules used:**

  - *Application modules:*

    `search(balboa)`, `search(db_names)`, `balboa_term_desc`, `search(utils)`.

  - *System library modules:*

    `sets`, `hiordlib`, `sort`, `lists`, `aggregates`.

  - *Internal (engine) modules:*

    `hiord_rt`, `arithmetic`, `atomic_basic`, `attributes`, `basic_props`, `basiccontrol`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

7.2 Documentation on exports (`match`)

match_sort_interval_list/9: PREDICATE

Usage: `match_sort_interval_list(SortMethod, Mode, Wanted,`

`IncPack, ExcPack, Ini, Len, MatchList, Total)`

- *Description:* This is the main procedure for performing a search expressed in the internal language. This procedure generates a list `MatchList` containing the resulting assemblies of a given query. A query is expressed in the internal query language, and is defined as follows:
  - `SortMethod` specifies the sorting method used for the query. Several values are allowed, as described in `sort_method/1`.
  - `Mode` selects the search type. Allowed values are described in `match_mode/1`.
  - `Wanted` is the list of capabilities to search. It must be provided to the matching engine as a list of description terms.
  - `IncPack` allows to specify that the results of the matching engine must contain a mandatory set of packages, represented by a list of package names. If there are no packages to include, `IncPack` must be an empty list.
  - In the same way, `ExcPack` specifies a (possibly empty) list of packages which should not appear in the resulting assemblies given by the matching engine.

  The assemblies provided by the matching engine that satisfy the query are sorted using the criteria specified in `SortMethod`, and returned in subsets of length `Len`, starting with the `Ini`th element of the complete results list (the first element of the list is numbered 0). In `Total` the total number of results found is returned.
- *Calls should, and exit will be compatible with:*

  `SortMethod` Weights assigned for the different sort options in the internal query language. (`sort_method/1`)

  `Mode` Matching modes available in the search engine. Available modes can be:
  - `all` returns all the assemblies which match the query.
  - `best(N)` selects the best `N` assemblies which satisfy the query.
  - `gen(N,M)` extends the matching of assemblies taking into account possible generalizations: `N` determines the maximum level of generalizations in preconditions (the description terms that a package requires), while `M` specifies the level of generalizations in postconditions (terms that a package provides).

(match_mode/1)

`Wanted` is a list. (list/1)

`IncPack` is a list. (list/1)

`ExcPack` is a list. (list/1)

`Ini` is an integer. (int/1)

`Len` is an integer. (int/1)

`MatchList` Sorted result of a search (match_sort_result/1)

`Total` is an integer. (int/1)

– *The following properties should hold at call time:*

`SortMethod` is currently ground (it contains no variables). (ground/1)

`Mode` is currently ground (it contains no variables). (ground/1)

`Wanted` is currently ground (it contains no variables). (ground/1)

`IncPack` is currently ground (it contains no variables). (ground/1)

`ExcPack` is currently ground (it contains no variables). (ground/1)

`Ini` is currently ground (it contains no variables). (ground/1)

`Len` is currently ground (it contains no variables). (ground/1)

– *The following properties hold upon exit:*

`SortMethod` is currently ground (it contains no variables). (ground/1)

`Mode` is currently ground (it contains no variables). (ground/1)

`Wanted` is currently ground (it contains no variables). (ground/1)

`IncPack` is currently ground (it contains no variables). (ground/1)

`ExcPack` is currently ground (it contains no variables). (ground/1)

`Ini` is currently ground (it contains no variables). (ground/1)

`Len` is currently ground (it contains no variables). (ground/1)

`MatchList` is currently ground (it contains no variables). (ground/1)

`Total` is currently ground (it contains no variables). (ground/1)

match_mode/1: REGTYPE

Usage: `match_mode(X)`

– *Description:* `X` Matching modes available in the search engine. Available modes can be:

* `all` returns all the assemblies which match the query.
* `best(N)` selects the best `N` assemblies which satisfy the query.
* `gen(N,M)` extends the matching of assemblies taking into account possible generalizations: `N` determines the maximum level of generalizations in preconditions (the description terms that a package requires), while `M` specifies the level of generalizations in postconditions (terms that a package provides).

match_sort_result/1: REGTYPE

```
match_sort_result(X) :-
        list(match_result_key,X).
```

Usage: `match_sort_result(X)`

- *Description:* `X` Sorted result of a search

sort_method/1: REGTYPE

```
sort_method(key_weights(P,U,T,C,B)) :-
        weight__package_number(P),
        weight__unsatisfied_deps(U),
        weight__tighter_assemblies(T),
        weight__capabilities(C),
        weight__best_capabilities_ratio(B).
```

Usage: `sort_method(X)`

- *Description:* `X` Weights assigned for the different sort options in the internal query language.

weight__package_number/1: REGTYPE

Usage: `weight_package_number(X)`

- *Description:* `X` Weight assigned to the number of packages provided by the matching engine.

weight__unsatisfied_deps/1: REGTYPE

Usage: `weight_unsatisfied_deps(X)`

– *Description:* X Weight assigned to the number of unsatisfied dependencies (less dependencies get higher weights).

weight__tighter_assemblies/1: REGTYPE

Usage: `weight__tighter_assemblies(X)`

– *Description:* X Weight assigned to the number of tighter assemblies (looser assemblies get higher weights).

weight__capabilities/1: REGTYPE

Usage: `weight__capabilities(X)`

– *Description:* X Weight for number of capabilities.

weight__best_capabilities_ratio/1: REGTYPE

Usage: `weight__best_capabilities_ratio(X)`

– *Description:* X Weight assigned to the ratio satisfied/unsatisfied capabilities.

package/1: REGTYPE

Usage: `package(X)`

– *Description:* X is a Amos package

capability/1: REGTYPE

Usage: `capability(X)`

– *Description:* X is a Amos capability

References

[CGC+04] M. Carro, J. M. Gomez, J. Correas, J. F. Morales, E. F. Mera, G. Puebla, D. Cabeza, F. Bueno, C. Daffara, and M. Hermenegildo. AMOS User's Manual. Technical Report CLIP4/2004.0, Technical University of Madrid, School of Computer Science, UPM, March 2004.

[CMM04] J. Correas, E. Mera, and J. F. Morales. Final matching engine. Technical Report CLIP8/2004.0, Computer Science School, Technical University of Madrid, School of Computer Science, UPM, May 2004. Deliverable D15 of the AMOS Project.

[GCM04] J. M. Gomez, M. Carro, and J. F. Morales. The external interface. Technical Report CLIP6/2004.0, Computer Science School, Technical University of Madrid, School of Computer Science, UPM, May 2004. Deliverable D12 of the AMOS Project.