PETINA :
PErsonalized Tourist INformation Advisor

Costas Halatsis   Maria Katzouraki*   Michael Hatzopoulos
Panagiotis Stamatopoulos*   Isambo Karali   Costas Mourlas
Manolis Gergatsoulis*   Evangelos Pelecanos

University of Athens
Department of Informatics
Panepistimiopolis, TYPA Buildings
157 71 - Athens

Abstract

PETINA is a PErsonalized Tourist INformation Advisor system about Greece aiming to help tourists to construct tours satisfying specified constraints. PETINA is going to be implemented in the ElipSys language, a parallel logic programming language under development. As it tries to solve a combinatorial searching problem, various ElipSys features, such as data-parallelism, data driven computation and constraint satisfaction techniques, can be exploited efficiently. Moreover, as it consults a large database containing tourist data, the external database connection facility of ElipSys is exploitable as well.

1 Introduction

In the following, we are going to present part of our contribution to the ESPRIT Project EP2025, European Declarative System (EDS). The project's duration is 4 years (1989-1992) and it is a collaboration between BULL (France), ICL (UK), SIEMENS (Germany) and ECRC (Europe). A number of other European companies and universities are also involved, including the Athens University as an associate partner of ECRC. The target of the EDS project is to design and implement both the hardware and the software for a parallel information server. The EDS machine is a message passing multiprocessor with distributed store. It consists of 4 to 256 Processing Elements (PEs), each one containing 64M to 2G bytes of memory. Three declarative programming paradigms are supported, namely database, Lisp and logic programming. ElipSys is the parallel logic programming subsystem that aims at the development of complex applications. OR-parallelism, data-parallelism, data driven

*NRCPS "Democritos", 153 10 - Aghia Paraskevi, Attiki

computation, constraint satisfaction through finite domains and an interface to the EDS database server are some of the main characteristics of the ElipSys language.

PETINA system [HKH+90] is a PErsonalized Tourist INformation Advisor about Greece. It takes as input user wishes about tour generation, expressed as constraints over visits' properties. Its output is tours satisfying the user's constraints, as sets of activities and as sets of events. Activities are considered to be the tourist's visits to various spots, while events are shows that the tourist may attend. The user is also allowed to ask for information about the activities and events, as well as about the geographical division of Greece. All the information PETINA requires to work is stored in a database. Management of this database is also supplied by the system.

PETINA is roughly based on existing prototypes developed at ECRC, namely TInA [Ing88] and TIA [Ing87b]. The application is going to be implemented in the ElipSys language [BCDR+89, BCDR+90, DRSX90].

In this paper, PETINA's database structure, the system's functional and structural specifications as well as an outline of the implementation of some significant parts of the system in Sepia [Sep90] and PEPSys [RR86] are presented. Sepia is a sequential logic programming system, though PEPSys is a parallel one that supports OR-parallelism and independent AND-parallelism, both developed at ECRC.

2 PETINA's Database

The system's database consists of the activities' and events' *instances* as well as the sites' ones. In the database, there also exist an activity and an event *tree*. The activity and event instances are linked to nodes of the corresponding trees. On the other hand, the site instances themselves compose a site tree.

Instances are characterized by their *attributes*. However, these attributes possibly also characterize nodes of the activity tree and event tree, though these nodes do not represent instances. Their values are considered to be the default ones of the instances linked either directly or not to these nodes, if no other value is specified explicitly.

The database is implemented as a Prolog database, though, when ported to ElipSys, instance information will be implemented using a relational database to be handled by ESQL [GVB+89]. ESQL is the EDS Extended SQL system, accessible by ElipSys through an interface.

2.1 Activity Information

The nodes of the activity tree represent activity categories. The tree's organization is based on interest hierarchy. The ElipSys implementation of the activity tree is carried out by the `activity_ako/2` predicate. The implementation of a part of the tree is:

```
activity_ako(package_tour(1),general_activity(1)).
activity_ako(activity(1),general_activity(1)).
  activity_ako(historical_place(1),activity(1)).
    activity_ako(ancient_history_place(1),historical_place(1)).
      activity_ako(historical_site(1),ancient_history_place(1)).
        activity_ako(wall(1),historical_site(1)).
        activity_ako(quarry(1),historical_site(1)).
```

A special node of the tree, namely the `package_tour(1)` node, includes instances which are predefined tours rather than single activity instances. These tours are considered predefined in the sense that they are available as package tours by Greek travel agencies.

Activity categories whose instances have various kinds of interest are represented by more than one tree nodes, denoted with the same keyword but with different indices. In this way we implement a graph idea with a tree structure. Activity instances can be linked to more than one nodes according to the types of interest they present or according to the categories they belong to. The links between the activity instances and the activity tree nodes are implemented by the `activity_isa/2` predicate, e.g.:

```
activity_isa(ainst00001,museum(1)).
activity_isa(ainst00001,museum(4)).
activity_isa(ainst00001,building(1)).
```

An activity instance is characterized by its attributes. These attributes are the following:

site: The site where the activity instance is. Its value is a node of the site tree.

denomination: The name of the instance.

duration: An estimate of the time needed to spend in the instance, in minutes.

cost: The entrance cost of the instance.

time period: The time period when it is possible to visit the instance.

closed days: The list of days when the instance is closed.

interest: The list whose elements express the amount of specific kinds of interest the instance presents. A kind is denoted in terms of a functor that corresponds to an activity tree node that is considered as interest node. For every kind of interest appearing in the list, the instance has to be linked, either directly or indirectly, to the corresponding node as well.

detail: Informative data about the instance.

The above are implemented by predicates of arity 2. To illustrate them, taking the example of `ainst00001`, their implementation is:

```
activity_site(ainst00001,'Corfou').
activity_denomination(ainst00001,'Archeological Museum of Corfou').
activity_duration(ainst00001,60).
activity_cost(ainst00001,200).
activity_time_period(ainst00001,from_to('01 Jan','31 Dec')).
activity_closed_days(ainst00001,['Saturday']).
activity_interest(ainst00001,[ancient_history_place(6),
                              culture(8),
                              modern_year_history_place(6)]).
activity_detail(ainst00001,'Archaic sculptures - Silver coins').
```

2.2 Event Information

The event tree nodes correspond to event categories. The tree's organization is based on event type hierarchy. The ElipSys implementation of the event tree is carried out by the `event_ako/2` predicate, e.g.:

```
event_ako(ancient_drama,cultural_event).
```

The structure of the event tree is simpler than the one of the activity tree. A pure tree idea is reflected, so there are no indexed nodes. The links between the event instances and the event tree nodes are implemented by the `event_isa/2` predicate, e.g.:

```
event_isa(einst00001,ancient_drama).
```

In contrast to the activity instances, event instances cannot be linked to more than one event tree nodes.

An event instance is characterized by its attributes. These attributes are the following:

site: The site of the spot where the event instance takes place. Its value is also a node of the site tree.

denomination: The name of the instance.

duration: The time needed to attend the instance.

cost: The entrance cost of the instance.

time period: The time period when the event is active.

interest: It represents the amount of interest of the instance. It is a single valued attribute, in contrast to the list valued corresponding attribute of activity instances.

takes place: The spot where the instance takes place.

series: The series of events where the instance is featured in.

detail: Informative data about the instance.

The above are implemented by predicates of arity 2 in a similar way to the activity instances' attributes, e.g.:

```
event_site(einst00001,'Epidaurus').
event_denomination(einst00001,'Oedipus at Colonus').
event_duration(einst00001,120).
event_cost(einst00001,1300).
event_time_period(einst00001,from_to('14 Jul 89','15 Jul 89')).
event_interest(einst00001,10).
event_takes_place(einst00001,'Ancient Theatre of Epidaurus').
event_series(einst00001,'Epidaurus Festival 1989').
event_detail(einst00001,'Performed by: National Theatre of Greece -
                         Written by: Sophocles - Directed by: A. Minotis').
```

2.3 Site Information

The nodes of the site tree are site instances. The tree hierarchy reflects a site inclusion relation. The site tree is implemented in ElipSys by the `belongs/2` predicate. A part of the site tree, as implemented in the system, is the following:

```
belongs('Greek islands','Greece').
  belongs('Aegean sea islands','Greek islands').
    belongs('Cyclades','Aegean sea islands').
      belongs('Paros','Cyclades').
        belongs('Paraika','Paros').
        belongs('Naoussa','Paros').
      belongs('Amorgos','Cyclades').
        belongs('Katapola','Amorgos').
    belongs('Dodecanese islands','Aegean sea islands').
      belongs('Rhodes','Dodecanese islands').
    belongs('Crete','Aegean sea islands').
```

The site attributes are the following:

type: The type of the site, i.e. island, village, city, province etc.

accommodation: An estimate of the site's accommodation capability.

approachability: An estimate of how easily the site can be reached from Athens by public transports.

entertainment: An estimate of the site's entertainment facility.

tour availability: An estimate of the site's capability in predefined tours.

eating facilities: An estimate of the site's eating facility.

detail: Informative data about the site.

The implementation of the above attributes, taking an example of a specific site, is as follows:

```
site_type('Dodecanese islands',island_group).
site_accommodation('Dodecanese islands',8).
site_approachability('Dodecanese islands',7).
site_entertainment('Dodecanese islands',8).
site_tour_availability('Dodecanese islands',8).
site_eating_facilities('Dodecanese islands',9).
site_detail('Dodecanese islands','SE Greece - Ancient and middle
                                  ages history - Hot and sunny -
                                  Heavily crowded during summertime').
```

In order to provide a means of accessing the names of the sites that satisfy a specific site information query, the following fact is sufficient:

```
site_denomination(Site,Site).
```

Another data structure which may be implemented in PETINA is a transportation graph. This could provide informative data about the types of transportation between sites, frequency of transportation, distances in Kms etc. We do not intend to allow the user to express constraints involving these data.

3 Functional Specifications

PETINA's Functional Specifications are shown in Figure 1. The user can be either a tourist or the database administrator.

The tourist is allowed to give to the system tour generation requests or information retrieval queries. The tour generation requests refer to either the activity or event tour generations. These tours are either sets of activity or event instances respectively. The system's answers to information retrieval queries possibly help the tourist to determine the constraints.

The administrator, apart from the above requests, may give to the system database administration commands. However, only these and the information retrieval queries are useful to his/her task.

In the current implementation, we consider an approach for the user's communication with the system through a formal language. However, in the future we intend to develop a more friendly interface.

3.1 Tour Generation

There are two kinds of tours the system produces. Consequently, there are two kinds of tour generation requests the user may express. The one concerns the activity tour generation and the other the event tour generation. In both cases, at the beginning of the request, the user has to give a time constraint concerning the time period when his/her visit is going to take place in order to avoid visiting spots that are inactive. The other part of the request is a set of activity or a set of event constraints that the solutions have to satisfy. A constraint may be either *simple* or *cross*. A simple constraint is satisfied in case a property of a set of instances holds. This property may refer to every instance of the set or to the entire set as a whole. A special property is the number of instances in a set. On the other hand, a cross constraint involves comparison between properties of two sets.

Examples:

1.
```
visit period is 2 Aug 89 - 2 Sep 89 &
site in ["Athens","Crete"] for activity &
number between [4,6] for activity &
number < 3 for museum with modern year history place interest &
cost < 100 for gallery where interest < 5 &
number = 1 for temple where denomination in ["Parthenon"] &
sum(cost) < 1000 for beach, spring &
max(duration) < 60 and interest > 6 for historical collection &
number for church > number for mosque.
```

   The last constraint of the request is a cross one, while all the other are simple.

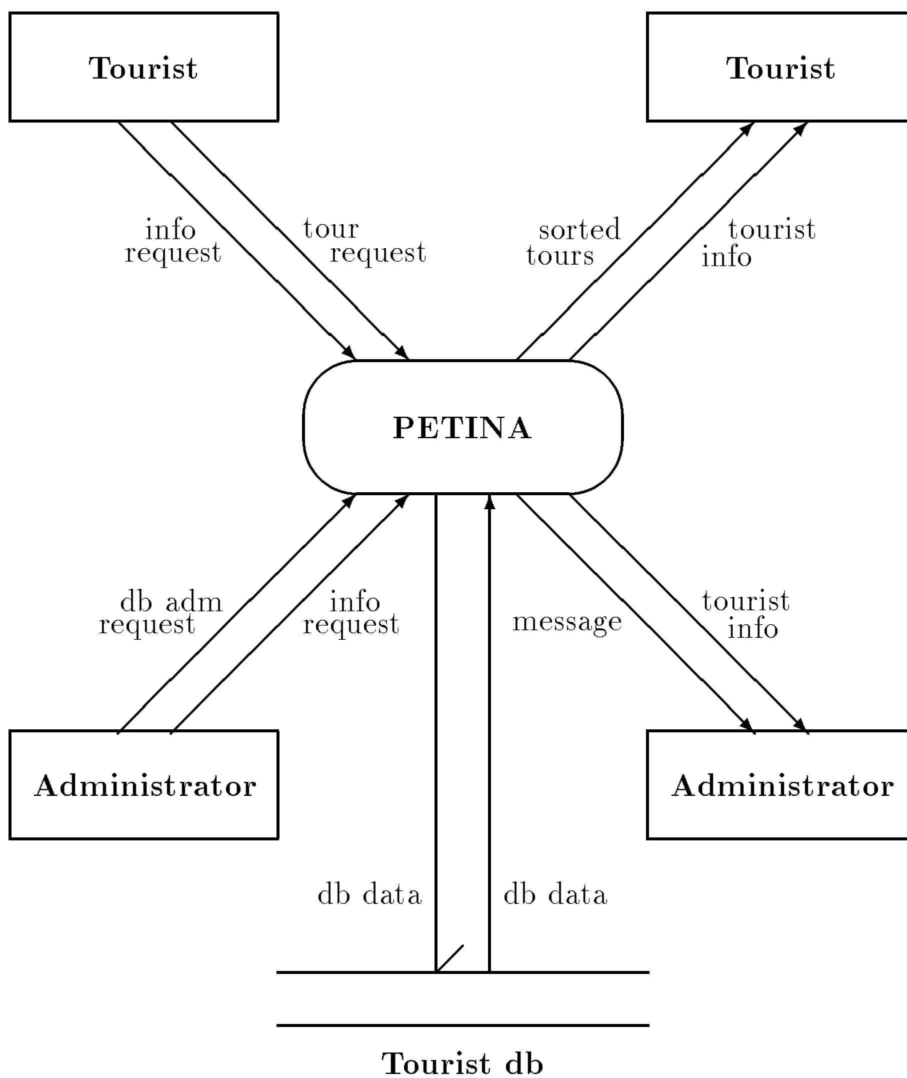

Figure 1: PETINA's Functional Specifications

2. ```
   visit period is 2 Jun 89 - 20 Jun 89 &
   site in ["Macedonia"] for event &
   site in ["Thessaloniki"] for festival movie &
   number = 4 for event &
   takes place in ["Philippi theatre"] for ancient drama &
   min(duration) > 90 or not avg(cost) between [250,350]
         for festival movie.
   ```

3.2 Information Retrieval

The system is able to supply information about activity or event instances and their attributes. Moreover, it can provide activity and event tree information as far as their structures are concerned. Finally, the user can be informed about the links between activity or event instances and the corresponding tree nodes.

Examples:

1. `cost for museum where site in ["Athens"].`
2. `avg(duration) for festival movie.`
3. `number for church, monastery.`
4. `kinds of cultural event.`
5. `whose kind is building.`
6. `instances of local celebration.`
7. `what activity is "Goulandris museum".`

Another facility the system provides is information about sites. This is considered very important as the user may not be familiar with the geography of Greece. The user may ask for information about site instances and their attributes. Furthermore, information about the site inclusion relation can be obtained.

Examples:

1. ```
   denomination in the area of "Aegean sea islands"
         where type = island and accommodation > 8.
   ```
2. `max(entertainment) in the area of "Crete" where type = city.`
3. `what is included in "Macedonia".`
4. `where is "Naoussa".`

3.3 Database Administration

We allow the administrator to perform three kinds of operations on the database, namely deletion, insertion or update of its contents.

He/she may delete either instances of the database or nodes of the activity or event tree.

Examples:

1. `delete instances for cathedral, mosque.`
2. `delete fair.`
3. ```
   delete in the area of "Amorgos" where type = town
          and accommodation < 6.
   ```

He/she may make insertions of either instances or tree nodes. New instances may be inserted, linking them with specified tree nodes. In addition, activity instances may be linked with other tree nodes as well, apart from the ones they were linked with so far. As far as trees are concerned, new nodes may be inserted under a specified node.

Examples:

1. ```
   insert denomination = "Naxos", type = island,
          accommodation = 8 at site "Cyclades".
   ```
2. ```
   share with building with architectural place interest instances
          for museum where denomination in ["Goulandris museum"].
   ```
3. ```
   insert palace at node historical site
          with ancient history place interest.
   ```

The administrator may update values of attributes both of instances and nodes. In addition, he/she may rename nodes of the trees or move nodes under another one. Finally, instances may be moved from the node they were linked with so far to another different one.

Examples:

1. `set default cost = 0 at node church.`
2. ```
   set interest = 10, duration = 150 for festival movie
          where denomination in ["Sweet gang"].
   ```
3. ```
   set entertainment = 8 in the area of "Ionian sea islands"
          where type = island.
   ```
4. `rename park to garden.`
5. `move to cultural event the node local celebration.`
6. `move to cultural event instances for festival movie.`

Integrity constraints are defined which the database has to satisfy after executions of database administration commands. These constraints concern the type of attribute values, their range etc. Moreover, a minimal set of attributes is defined as compulsory to be provided in case a new instance is inserted.

4 Structural Specifications

PETINA (Figure 2) consists of various modules to carry out its different functions. These modules are:

- User Interface
- Tour Generation Engine
- Information Retrieval Engine
- Database Administration Engine

The User Interface module is responsible for the user-system communication. The Tour Generation Engine, the most important module of the system, generates activity and event tours satisfying the user's constraints. The Information Retrieval Engine supplies the information the user asked for. Finally, the Database Administration Engine manages the database contents.

From now on, we concentrate on the Tour Generation Engine module, as it carries out the main function of PETINA and its significant complexity requires the exploitation of a variety of ElipSys features.

The Tour Generation Engine (Figure 3) consists of the following modules:

- Tour Generation Parser
- Domains Creator
- Configurator
- Database Filter
- Tour Generator
- Tour Evaluator

The Tour Generation Parser transforms the request into structures suitable for further processing. The Domains Creator partitions the appropriate tree (activity or event) into domains in such a way that no two domains have the same set of constraints applied to them. The Configurator produces all possible configurations for the solutions, taking into account constraints that refer to desired number of instances that have a specific property. The Database Filter determines the candidate instances for the solution by rejecting ones that do not satisfy the constraints that are applied to individual instances. The Tour Generator produces all tours, according to the configurations, that satisfy the user's constraints. Finally, the Tour Evaluator arranges the generated tours taking into account some criteria, currently the average interest.

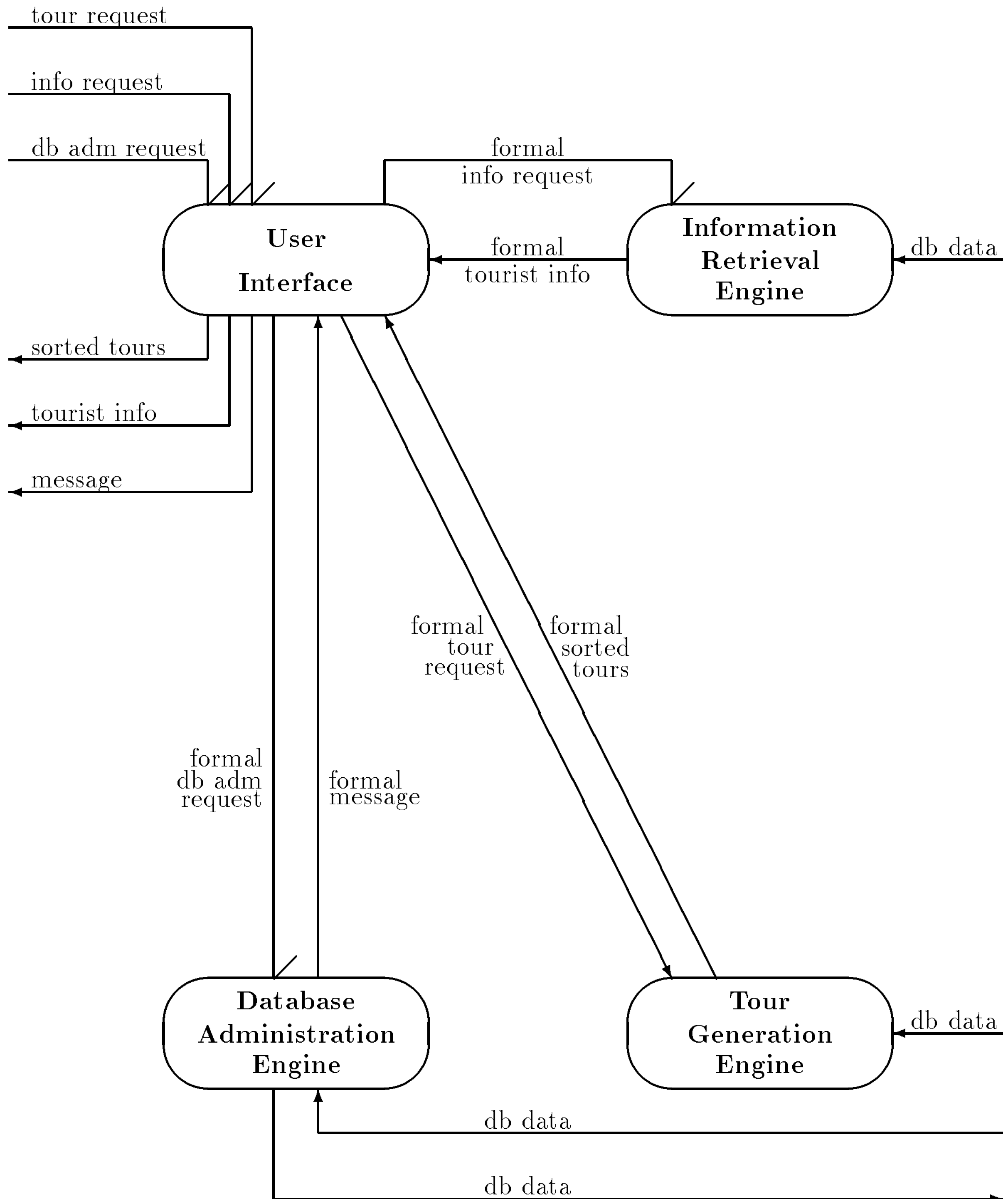

Figure 2: The PETINA System

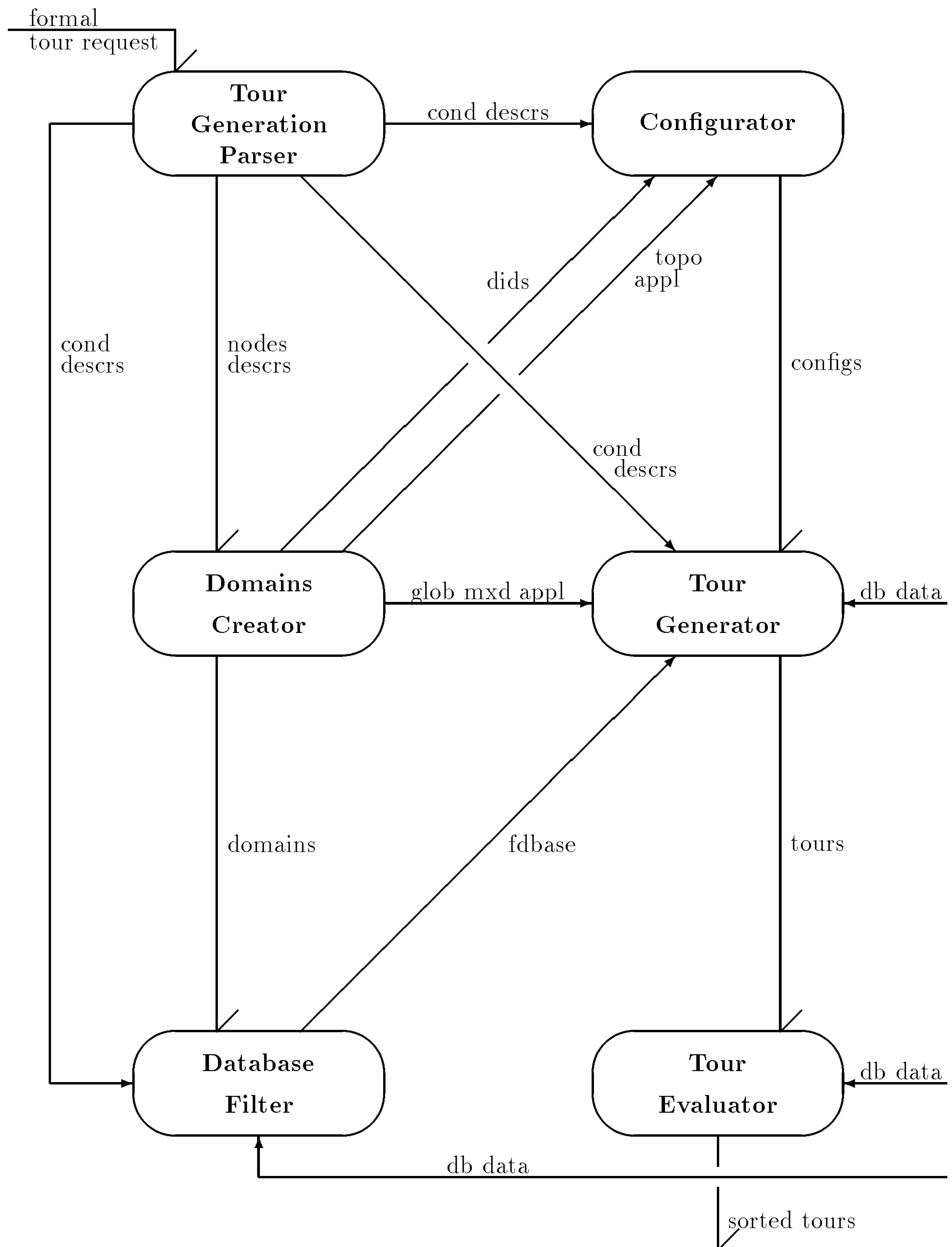

Figure 3: The Tour Generation Engine module

5 Implementation Issues

In this paragraph, we discuss about the implementation of the Tour Generation Engine. The sequential implementation in Sepia [Sep90] was carried out in such a way that eventually will be ported in ElipSys [BCDR+89, BCDR+90, DRSX90] with the least modifications. The Sepia 3.0 version was used running on a SUN 3/60 workstation under SunOS 4.0.

In order to achieve a better execution time, the *delay mechanism* of Sepia was used in various parts of the system. This mechanism introduces a data driven reduction rule on top of the depth-first left-to-right execution strategy of Prolog. Using this mechanism, there is a possibility to modify the reduction order of goals according to data dependencies. This is achieved by declaring some procedures to delay execution if a condition holds. This mechanism is offered in ElipSys as well [Rat89].

Another feature of ElipSys that improves the system's performance is *constraint satisfaction techniques* [vH89]. The use of these techniques leads to a priori pruning of the search space, thereby resulting in more optimized execution. Constraints are used by the programmer to express relationships between objects of finite domains. These constraints reduce the search space in such a way that they hold at any moment as the reduction process proceeds.

Finally, we profit a lot from the ElipSys parallelism. More precisely, a special kind of it, *data-parallelism* [Heu89], is suitable for our case. Data-parallelism stands for the concurrent treatment of the elements of a set of data. This kind of parallelism is supported in ElipSys by a set of built-in predicates. In order to experiment and measure the potential parallelism, we implemented part of the system in PEPSys [RR86] as well, using the COKE tool [Ing87a, HI89]. As data-parallelism is not supported by PEPSys, we simulated it by defining the appropriate predicates and declaring them to be executed in parallel.

In the following, we highlight the most interesting parts of the implementation of the modules the Tour Generation Engine consists of.

5.1 Tour Generation Parser

The Tour Generation Parser takes as input a formal tour request and transforms it into a form suitable for further processing using a Definite Clause Grammar (DCG) [PW80].

The condition of a simple constraint may be a complicated logical expression containing the operators "and", "or" and "not". This expression is transformed into conjunctive normal form, producing in this way a set of constraints that involve conditions which do not contain the operator "and". We also eliminate the operator "not" by inversing the condition relation. Thus, we end up with a set of simple constraints that each one involves a logical expression which contains at most the operator "or". After this process, in case all the conditions of a simple constraint apply to individual activity instances, the constraint is named *local*. If all of them apply to an entire set of instances, the constraint is named *simple global*. However, if they refer to numbers of the instances in a set, the constraint is named *simple topological*. Finally, if none of the above holds, the constrained is named *mixed*.

On the other hand, cross constraints do not require the previous processing and refinement, since they represent just a binary relation between properties of two sets. They are distinguished into *cross global* and *cross topological*, depending on the kind of the properties.

5.2 Domains Creator

The Domains Creator partitions the activity or event tree into *domains* relating every domain with the set of constraints that applies to it. This partitioning is based on global and topological constraints, both simple and cross, as well as on the mixed ones. Each domain is further partitioned into *fine domains* according to the local constraints. The above process is carried out in two passes.

5.3 Configurator

The Configurator produces all possible *configurations* of the tours. This module, taking into account the topological constraints, both simple and cross, generates and solves a system of linear equalities and inequalities. A solution to this system, that is a configuration, represents acceptable numbers of instances per domain in a tour satisfying the user's constraints.

A system of linear equalities and inequalities can be solved efficiently using constraint satisfaction techniques. ElipSys supplies this facility. However, Sepia 3.0 doesn't, so we exploited the delay mechanism of the language, though this mechanism, in this case, is not as efficient as the former one.

5.4 Database Filter

The Database Filter consults the database and selects the instances that satisfy the local constraints. Instances are selected for every node refined by its "where" property to build the instances of a fine domain. Then, such sets are structured to form the instances of a domain, leading finally to the composition of the *filtered database*.

For this module, a parallel version was implemented as well. Parallelism was exploited in three levels, the concurrent processing of domains, fine domains and nodes refined by their "where" properties. The kind of parallelism encountered is data-parallelism. For a demo request, the speedup achieved was 6.66 ni/et (number of inferences / execution time).

5.5 Tour Generator

The Tour Generator is the most important part of PETINA. It is the module where the actual tours are constructed. It selects instances from the filtered database to fill the slots in the configurations. Thus, a candidate tour is constructed. This tour is accepted, if it satisfies the simple global, cross global and mixed constraints.

The output of the Tour Generator is a list of lists of instances. Each list of instances represents an acceptable tour.

The method used for the construction of tours is test-and-generate implemented using the delay mechanism of Sepia. Although this mechanism may be also used in ElipSys, constraint satisfaction techniques may lead to a better performance.

The main source of parallelism of the whole system exists in this module. For this reason, we also implemented this module in PEPSys. There are two parts where parallelism can be efficiently exploited. Firstly, we process all possible configurations concurrently and secondly, and mostly, we select all possible instances to build a subtour for the corresponding domain in parallel. In both cases, the kind of parallelism is again data-parallelism. For the demo request, the speedup achieved was 141.61 ni/et. The main source of this speedup was due to the parallel selection of instances.

5.6 Tour Evaluator

The Tour Evaluator takes as input the tours produced by the Tour Generator and sorts them in descending order according to their average interest. In the near future, better evaluation criteria may be taken into account.

6 Conclusions

In the above we presented PETINA's database structure, the system's functional and structural specifications and an outline of the implementation of the most significant part of the system.

PETINA is a PErsonalized Tourist INformation Advisor about Greece consulting a large database that contains tourist data. The application is suitable to be developed in a parallel environment, as it tries to solve a combinatorial searching problem.

The system is intended to be implemented in the ElipSys language, a parallel logic programming language under development, profiting from its parallelism as well as other features it offers. The most important and cumbersome part of the system has been already implemented in a sequential logic programming system, Sepia. Parts of it, where parallelism can be efficiently exploited, were also implemented in PEPSys, a parallel logic programming language. The exploitable parallelism is data-parallelism that ElipSys supports. In the current implementation the delay mechanism of Sepia was found to be indispensable. Such a mechanism is provided by ElipSys as well. In addition, constraint satisfaction techniques, also supported by ElipSys, may dramatically improve PETINA's performance.

Taking into account the above, we believe that although we deal with a tourist database for a whole country and the search space seems to increase considerably, ElipSys features will help to overcome problems due to the complexity of the algorithms needed.

References

[BCDR+89] U. Baron, A. Cheese, S. Delgado-Rannauro, P. Heuzé, M.-B. Ibáñez-Espiga, and M. Ratcliffe. A first definition of the ElipSys logic programming language. Technical Report Elipsys/005e, ECRC, September 1989.

[BCDR+90] U. Baron, A. Cheese, S. Delgado-Rannauro, P. Heuzé, M.-B. Ibáñez-Espiga, and M. Ratcliffe. The ElipSys logic programming language. Technical Report CA-53, ECRC, February 1990.

[DRSX90] S. Delgado-Rannauro, K. Schuerman, and J. Xu. The ElipSys computational model. Technical Report CA-51, ECRC, February 1990.

[GVB+89] G. Gardarin, P. Valduriez, M. Bèrard, L. Chen, O. Gerbe, M. Lopez, and J. Mondelli. ESQL: An Extended SQL with Object Oriented and Deductive Capabilities. Project Deliverable EDS.DD.11B.0910, INFOSYS, December 1989.

[Heu89] P. Heuzé. Using Data-Parallelism in the ElipSys. Internal Report ElipSys-003, ECRC, June 1989.

[HI89] P. Heuzé and B. Ing. COKE: User manual 1.0. Internal report, ECRC, February 1989.

[HKH+90] C. Halatsis, M. Katzouraki, M. Hatzopoulos, P. Stamatopoulos, I. Karali, C. Mourlas, M. Gergatsoulis, and E. Pelecanos. PETINA: PErsonalized Tourist INformation Advisor—Specifications. Project Deliverable EDS.WP.9E.A003, University of Athens, January 1990.

[Ing87a] B. Ing. COKE—An analysis tool for PEPSys programmes. Internal Report 23, ECRC, October 1987.

[Ing87b] B. Ing. Tourist information advisor: A case study of an application in PEPSys. Internal Report PEPSys/15, ECRC, April 1987.

[Ing88] B. Ing. Tourist information advisor—A case study of an application in PEPSys—Final report. Internal Report PEPSys-32, ECRC, September 1988.

[PW80] F. Pereira and D. Warren. Definite clause grammars for language analysis—A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231–278, 1980.

[Rat89] M. Ratcliffe. On the use of the delay in ElipSys Prolog. Technical Report elipsys/001, ECRC, June 1989.

[RR86] M. Ratcliffe and P. Robert. PEPSy: A Prolog for parallel processing. Technical Report CA-17, ECRC, April 1986.

[Sep90] *Sepia 3.0 User Manual*, June 1990.

[vH89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.