ROMEO: An Ontology-Based Multi-Agent Architecture for Online Information Retrieval

Dmitri Soshnikov, Irina Krasteleva
Department of Numerical Mathematics and Programming,
Moscow Aviation Institute (State Technical University)
Moscow, Russia
dmitri@soshnikov.com, irina@krasteleva.com

Abstract

This paper describes an approach to path-finding in the intelligent graphs, with vertices being intelligent agents. A possible implementation of this approach is described, based on logical inference in distributed frame hierarchy. Presented approach can be used for implementing distributed intelligent information systems that include automatic navigation and path generation in hypertext, which can be used, for example in distance education, as well as for organizing intelligent web catalogues with flexible ontology-based information retrieval.

1. Introduction

One of the major problems in information technologies nowadays is the growing amounts of information accumulated in large weakly-structured repositories such as the Internet. Thus a lot of research is going on in the area of information retrieval and search, and in most of the solutions some elements of artificial intelligence are present.

Many efforts are related to more structured unified information presentation on the WWW. The most perspective standard is Semantic Web [Semantic Web] proposed by the W3C, which allows to formulate ontology-based annotations for web resources and use them in the intelligent search process to find relevant information.

In SemanticWeb, as well as in many other projects, the emphasis is done on providing static description of resources, and applying some intelligent search algorithm to find relevant ones. At present, static description methods are rather well-developed, while there is still lack of efficient algorithms to handle large amounts of distributed knowledge.

However, there is another approach focused on providing dynamic descriptions in the form of local knowledgebases that contain dynamic knowledge in the form of production rules, and can provide references to recommended resources (including other knowledgebases to consult) upon request. Algorithms for logical inference in the production knowledgebases are well-known and tools already exist that can be used to implement logical inference in distributed environments [Soshnikov 2002, Davison 1998]. Some attempts to implement Internet search based on inference in distributed knowledgebase-like descriptions has already been done [Sizikov 2002].

The main disadvantage of this approach is the difficulty to implement a knowledgebase to describe rapidly-changing internet resources in a given problem domain. To overcome this difficulty, it is suggested to use predeveloped ontologies already containing universal dynamic knowledge in multiple problem domains, and thus reduce the task of describing individual resources to inheriting most of the knowledge from certain ontologies. Two types of ontologies are introduced: architectural ontologies that define different types of nodes in intelligent knowledgebase graph (eg., redirector, catalog, wrapper, keyword-based search engine, etc.), and domain ontologies.

Required resources are obtained in the process of online traversal of the collection of distributed knowledge-bases, that form a structure that we would refer to as intelligent graph. In each vertex, knowledge-base consultation occurs, with some questions possibly asked to the user. As a result of this consultation problem state is changed (new facts, either obtained from the user or derived, are added), and resource queue is altered, i.e. new resources to visit are suggested, which may be either resources of direct interest to the user, or further knowledgebases to consult.

Further on, we may impose some constraints on the order in which resources are visited, i.e. require that certain resources are visited prior to another ones. This would allow us to obtain a path in the resource collection that suits certain user needs. For example, in such a way we may obtain a navigation path among a series of related web resources, which turns out to be extremely useful in such areas as distance education.

In this paper, we first introduce the concept of intelligent graph and describe ad-hoc heuristic algorithm that can be used to implement search in such a graph. Then we describe an architecture of ROMEO toolkit (Retrieval-Oriented Multiagent Environment/Ontology) based on distributed frame hierarchy, that can be used to implement presented ideas of intelligent information retrieval. Finally, potential applications of the presented approach are discussed.

2. Path-finding in Intelligent Graphs

2.1. Intelligent Graphs

In many practical situations it is convenient to deal with graphs in which arcs are not statically defined, but rather depend on some parameters. For example, an intelligent web site catalog can present only a subset of available hyperlinks to the user, depending on his preferences, thus creating a dynamic hyperlink graph. Further on, user preferences can be altered during traversal of the hypertext graph, thus the set of arcs presented to the user would be dynamically determined in each vertex in the process of traversal. We would reflect this idea in the following definition.

Definition 1. By **State-dependent dynamic graph** we would mean $\mathfrak{G} = \langle \{X_i\}_{i=1}^N, \{\langle u_j, v_j, p_j\rangle\}_{j=1}^M \rangle$, where $\{X_i\}$ — a set of vertices, $u_j, v_j \in \mathbf{X} = \{X_i\}$, $p_j : \mathcal{S} \to \{\mathbf{true}, \mathbf{false}\}$ — set of arc activation functions for each arc defined on some abstract set of states $\mathcal{S}$.

For each $S \in \mathcal{S}$, a dynamic graph $\mathfrak{G}(S)$ becomes a normal directed graph $G = \langle \{X_i\}, U\rangle$, where $U = \{\langle u_j, v_j\rangle | p_j(S) = \mathbf{true}\}$.

We would consider state-dependent dynamic graphs in terms of path finding, i.e. finding a certain finite sequence of vertices $\langle X_0, \ldots, X_n\rangle$ starting from a certain point X_0, where final vertex X_n satisfies some condition $C(X_n)$. In practice we would also need to impose some constraints on the required path, in the form of order relation $\sqsubset$ defined on $\mathbf{X}$, which would also depend on the state S. We would also consider state changes during the search process, i.e. while traversing the graph some state change functions would be applied corresponding to each vertex. This leads us to the following definition:

Definition 2. A **a state-dependent constrained graph** $\mathfrak{G} = \langle \{X_i, T_i\}, \langle u_j, v_j, p_j\rangle, \sqsubset_S\rangle$, where $T_i : \mathcal{S} \to \mathcal{S}$ — state transformation functions, $\sqsubset_S$ — constraint order relation dependant on the current state. We also assume that $\mathcal{S}$ is a partially ordered set (or a lattice), and that relations p_j and $\sqsubset_S$ are monotonous, i.e. $\forall\, S_1 \sqsubseteq S_2 \in \mathcal{S}$ it holds that

$$\forall j\ p_j(S_1) \supset p_j(S_2) \qquad \forall u, v \in \mathbf{X}\ (u \sqsubset_{S_1} v) \supset (u \sqsubset_{S_2} v)$$

and that T_i are monotonous in the traditional sense $(\forall i\ \forall S \in \mathcal{S}\ S \sqsubseteq T_i(S))$.

While performing search in such a graph, the path would depend also on the starting state $S_0 \in \mathcal{S}$. Thus, each path $X_0 \to \ldots \to X_n$ also corresponds to a path $S_0 \sqsubseteq \mathcal{S}_1 \sqsubseteq \ldots \sqsubseteq S_n$ in the set of states. To account for this, we will make the following

Definition 3. A path $\langle X_0, S_0\rangle \to \ldots \to \langle X_n, S_n\rangle$ will be called **valid**, if $\forall i\ \langle X_i, X_{i+1}, p_j\rangle \in U$ and $p_j(S_i) =$ **true** (i.e. all arcs in the path are valid), and $X_i \sqsubseteq_{S_n} X_j \supset (i \leq j)$ (i.e. all final constraints are satisfied). We may also require monotonicity, i.e. $S_{i+1} \sqsubseteq T_i(S_i)$, and call it **monotonously valid path**.

In practive, when implementing search algorithms, the validity relation is difficult to accomplish, since vertices in a valid path have to satisfy all constraints, even the ones which may appear only on a later stage. However, we may consider a special kind of **separable constaint relation**, where $\sqsubset_{S_{i+1}} = \sqsubset_{S_i} \cup \sqsubset_i$, and $\sqsubset_i \subseteq \{\langle u, X_i\rangle, u \in \mathbf{X}\}$ (i.e. constraints are added in order of vertice traversal, and on a latter stages constraints on already traversed vertices are not introduced). In this case the validity relation can be written in a weaker form $X_i \sqsubseteq_{S_j} X_j \supset (i \leq j)$.

One practical application of presented definitions is in multi-agent systems oriented towards information retrieval and hypertext navigation planning. Consider a connected agent society, where each agent can reason in some common terms, and knows about a certain set of other agents that it can recommend for further consultation. In this case agents form dynamic graph, and finding a path satisfying certain constraints corresponds to performing distributed consultation, in the process of which required navigation path or desired resource are found. State $\mathcal{S}$ in this case would correspond to the common environment of all agents in the society, and it would obey all monotonic requirements if monotonic inference is used throughout the society. For more strict mathematical modeling of such societies, $\mathcal{S}$ can be considered to be a set of state functions $\mathbb{W}$, as defined in [Soshnikov 2002].

2.2. Path-finding Algorithms

For searching state-based constrained graphs, two approaches can be applied. First, most of the classical path-finding algorithms can be used, adding restrictions to satisfy constraints. As an example, path

Algorithm 1: Queue-based path-finding algorithm for separable constrained dynamic graph.

```
Input: Separable constrained dynamic graph $\mathfrak{G} = \langle\{X_i, T_i\}, U, \sqsubset_S\rangle$, initial search state $S_0$, initial point $X_0$, search criteria $C$.
Output: A set of all valid paths $R = \{\langle\langle X_0, S_0\rangle, \ldots, \langle X_n, S_n\rangle\rangle\}$
QueueSearch($\mathfrak{G}$,$X_0$,$S_0$)
$R \leftarrow \varnothing$
$Q \leftarrow \{\langle X_0, S_0\rangle\}$
while $Q \neq \varnothing$ do
    select $P \in Q$ (first, last, best, etc.)
    $Q \leftarrow Q \setminus \{P\}$
    $\langle X_l, S_l\rangle \leftarrow \mathrm{last}(P)$
    if $X_l$ satisfies $C$ then
        $R \leftarrow R \cup \{P\}$
        continue
    $S \leftarrow \{\langle x, T_x(S_l)\rangle | \langle X_l, x, p\rangle \in U, p(s) = \mathbf{true}\}$
    foreach $\langle x, s\rangle \in S$
        if $[P|\langle x, s\rangle]$ satisfies $\sqsubset_s$ (or $\sqsubset_x$) then
            $Q \leftarrow Q \cup \{[P|\langle x, s\rangle]\}$
```

queue keeping algorithm (which corresponds to either of depth-first, breadth-first of best-first search) modified for constraint satisfaction is presented in Algorithm 1.

Alternatively, constraints can be used not only to filter out invalid paths during the search process, but rather to select the search direction during path extension. For example, if, during the search process, we move from X_k to a certain vertex X_i, and constraints indicate that X_{j_1} and X_{j_2} has to be visited prior to X_i, then we proceed to X_{j_1} instead, placing X_{j_2} and X_i into the queue. In this example, however, a path from X_k to X_{j_1} may not exist — which should cause backtracking in the search algorithm. In practice, however, constraints indicate that corresponding vertex "knows" of the preceding one, and thus a path between them also exists, i.e.

$$X_i \sqsubset_S X_j \supset \langle X_i, X_j, p_j\rangle \in U \wedge p_j(s) = \mathbf{true} \qquad (1)$$

Those considerations lead us to a family of *ad hoc* heuristic path finding algorithms, where we are mostly concerned with constraint satisfaction and final search condition, rather than with the validity of path. Those algorithms would, however, give the valid path, given conditions (1) on the functions p_j and constraints. An example of such an algorithm is provided as Algorithm 2.

3. ROMEO: A Toolkit for Intelligent Search

3.1. Principles of Intelligent Search in Dynamic Resource Descriptions

Formalism presented above can be used to describe the following situation. Suppose there is a number of

Algorithm 2: Ad hoc heuristic path finding algorithm.

```
Input: Separable constrained dynamic graph $\mathfrak{G} = \langle\{X_i, T_i\}, U, \sqsubset_S\rangle$, initial search state $S_0$, initial point $X_0$, search criteria $C$.
Output: A path $P = \langle\langle X_0, S_0\rangle, \ldots, \langle X_n, S_n\rangle\rangle$
HeurSearch($\mathfrak{G}$,$X_0$,$S_0$)
$P \leftarrow \langle\langle X_0, S_0\rangle\rangle$
$Q \leftarrow \varnothing$
$R \leftarrow \varnothing$
while last($P$) does not satisfy $C$ do
    $\langle x_l, s_l\rangle \leftarrow \mathrm{last}(P)$
    if $Q \neq \varnothing$ then
        select $x \in Q$
        $Q \leftarrow Q \setminus \{x\}$
    else if $\{\xi | \langle x_l, \xi, p\rangle \in U, p(s_l) = \mathbf{true}\} \neq \varnothing$
        $S \leftarrow \{\xi | \langle x_l, \xi, p\rangle \in U, p(s_l) = \mathbf{true}\}$
        select $x \in S$
        $R \leftarrow R \cup (S \setminus \{x\})$
    else if $R \neq \varnothing$
        select $x \in R$
        $R \leftarrow R \setminus \{x\}$
    else
        break
    $s \leftarrow T_x(s_l)$
    $\Omega = \{\xi | \xi \sqsubset_s x\}$
    if $\Omega = \varnothing$ then
        $P = [P|\langle x, s\rangle]$
    else
        $Q = Q \cup \Omega \cup \{x\}$
```

resources, and we want to provide an automatic way for the user to select desired resource or path within those resources based on some preferences. This can be accomplished by associating with each resource a dynamic knowledgebase, that can determine whether resource is applicable to user preferences, and also recommend which resources should also be visited, either as a prerequisite, or as a relevant information source.

In this case, the resources would form a constrained state-based graph, where the state corresponds to the common working memory (or a blackboard) used by all individual expert systems. User preferences would determine the initial state S_0, and the initial point of search X_0 would be some well-known resource (search engine). In the search process resource graph would be traversed, and knowledgebase consultation performed at each node, resulting in enriching state by newly derived facts (which corresponds to applying monotonous function $T_x(\cdot)$).

During the consultation, some further questions may also be asked to the user to make decisions more precise. In this case, user answers are also stored in the state.

Constraints will be used to ensure that resources are traversed in the "correct" order, i.e. certain consultations are applied before other ones. As a result, either resources that satisfy user preferences, or a path

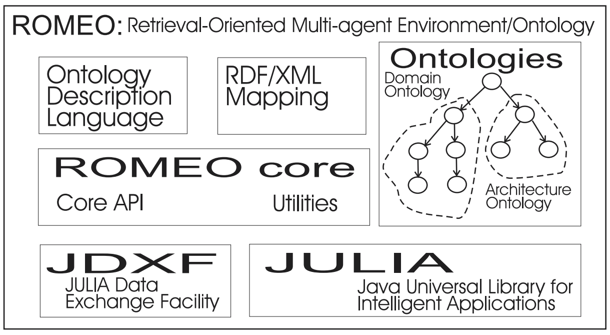
ROMEO: Retrieval-Oriented Multi-agent Environment/Ontology
Ontology Description Language
RDF/XML Mapping
Ontologies
Domain Ontology
Architecture Ontology
ROMEO core
Core API
Utilities
JDXF
JULIA Data Exchange Facility
JULIA
Java Universal Library for Intelligent Applications

Figure 1: Architecture of the ROMEO toolkit

in the resource graph can be obtained.

In order to ensure standardization and compatibility between knowledgebases describing different resources, they have to be formulated and reason about some common terms, that are defined by ontologies. In particular, two types of ontologies are involved: **architecture ontology**, that defines basic structures for describing resources, as well as templates for different types of resources such as catalog, forwarder, search engine wrapper, etc., and **domain ontologies**, that provide some domain-specific knowledge (eg., ontology for entertainment sites, which may contain specifications of different types of entertainment sites, templates for typical user preferences, statistical information for different age groups, etc.).

For implementation, it is also convenient to use such architecture where it is possible to define dynamic knowledge as part of the ontology, and use this domain knowledge directly in the reasoning process of each expert system. An architecture of distributed frame hierarchy described in [Soshnikov 2002] is very well-suited for this purpose, as it uses uniform knowledge representation throughout the whole system, which can directly be used in the process of distributed logical inference by a series of knowledgebases located on different network nodes. Furthermore, two modes of interoperability for distributed operation (remote invocation and inclusion) allow to use the system on different types of sites (both on active and passive servers), either via CORBA, RMI or simple HTTP protocols, in different configurations (with central or distributed inference), etc.

3.2. Toolkit Architecture

The overall architecture of the toolkit based on the described principles is shown on Fig.1.

The basis of the architecture is JULIA toolkit [Soshnikov 2002] that provides basic functionality for implementing reasoning in distributed frame hierarchy, which is enhanced by adding specific libraries for arbitrary data exchange between nodes and for accessing other HTTP-based resources, such as traditional search engines. On top of it, there are ROMEO core API and utilities, which implement the following functionality:

- **User interface**, either in the form of stand-alone application, UI agent, web interface or e-mail gateway.
- **Parsers** that convert XML or RDF/XML-based descriptions and knowledgebases into JULIA internal format.
- **Agents** that run on each node in the resource graph and perform resource description parsing and logical inference as required.

Important part of ROMEO toolkit are architectural and domain-specific ontologies, formulated in one of the supported ontology-description languages.

Relation of resource graph and different ontologies is demonstrated on Fig.2. Here two levels of ontological descriptions are shown: upper ontologies, and resource descriptions, which actually form the resource graph. Actual descriptions are also formulated in ontology description language, and are in fact a collection of frames inherited from some concepts in upper-level ontologies, or (possibly) from other resource descriptions. Typically, each resource description should directly or indirectly inherit its resource type from architectural ontology, and some domain knowledge from domain ontology. If we are describing typical resource, domain ontology should provide enough knowledge, and thus actual description would be as simple as static description.

In accordance to JULIA architecture of distributed frame hierarchy, there could be two ways of interoperability between nodes in resource graph and ontology descriptions: **invocation** or **inclusion**. With invocation, whenever a reference to a remote resource or ontology is made, a remote call is executed, and logical inference is transferred to another node. This require another node to be active, i.e. be able to serve as logical inference server and accept requests using some remote invocation protocol (CORBA, RMI, etc.). On the contrary, inclusion can function on any HTTP-capable server, and is basically provides required knowledge in some internal form, allowing to download parts of ontology hierarchy, instantiate it locally and use for logical inference. Two forms of interoperability provide almost identical functionality, and can also be combined to create complex interoperability schemes. For example, end resources can be described by passive descriptions that are located on HTTP server alongside with HTML pages, while some central server can be used to collect information from those resources, reason about it, and return links to

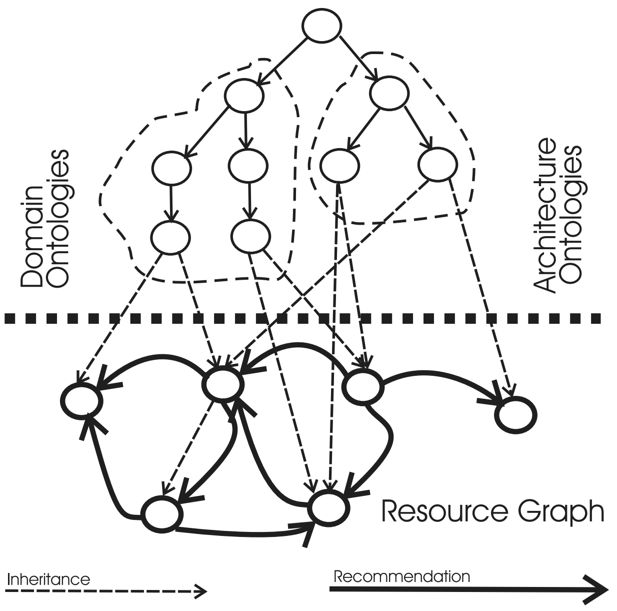

Figure 2: Ontologies and resource graph

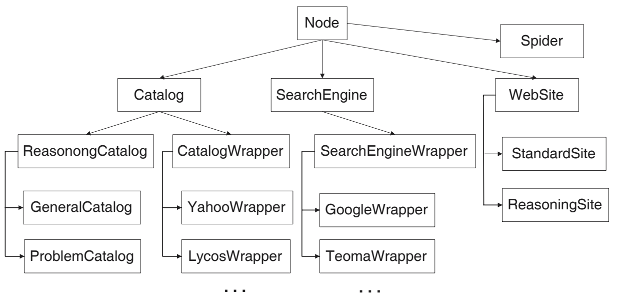

Figure 3: Architecture ontology

the user that is using thin client with limited reasoning capabilities (for example, a cell phone).

In principle, the same basic architecture can be implemented on any other platform that support reasoning in distributed knowledge repositories, for example on LogicWeb [Davison 1998]. JULIA toolkit, however, has an advantage of easy integration with procedural code written in Java, which makes it easier to expose certain parts or ROMEO core API as frames and part of ontology.

3.3. Ontology structure

Architecture ontology defines the basic structure of resource description node, and different types of nodes according to their general function. The following types of nodes can be identified:

WebSite A simple node describing one WWW resource. Such description is represented as a frame with specific slot that defines suitability of the resource to the user. Individual resource descriptions inherit from this graph and can redefine the suitability slot according to their content. Also, different sub-frames are already defined in the ontology, that evaluate suitability either according to keyword set, by domain ontology rules, etc. With those more specific concepts individual resource descriptions can be as simple as static descriptions, with all logic inherited from ontology.

Catalog A node that contains references to other nodes and can recommend certain resources based on user preferences. Again, different pre-defined types of catalogs exist, some of them also being a wrappers around existing Internet catalogs such as Yahoo.

ReasonongCatalog A ROMEO resource that contains knowledge base with intelligent references to other resources. We can distinguish between domain catalogs (that "know" of resources in certain problem domain), or more general catalogs, that resemble traditional categorial portals.

Wrapper A node that uses traditional search engine (SearchEngineWrapper) or catalog (CatalogWrapper) to retrieve web sites and present them as some form of resource descriptions. Each wrapper is unique for each search engine: GoogleWrapper, TeomaWrapper etc.

Retriever/Spider A node that collects information on other ROMEO resources by either crawling the web, using traditional search engines, or some peer-to-peer technique (for example, collecting all resources from the retrieval queue passed through it).

Domain ontologies provide knowledge specific to problem domains. For example, there could be an ontology developed to describe entertainment sites, which would provide classification of different types of entertainment resources, rules describing suitability of entertainment types to age groups, etc. Then, a certain entertainment site may be described by just inheriting its description from concept corresponding to a specific type of entertainment, and an entertainment catalog would use this description as well as rules from upper ontology to find this resource to be suitable to specified age group. Specification of age groups should also be standardized throughout all descriptions by making it a part of upper domain ontology for commonsense knowledge.

4. Applications

4.1. Search in the Internet

Presented architecture can be used for finding resources given certain flexible set of preferences or in the process of user consultation. The problem, however, is in producing resource descriptions for rapidly changing hypertext structure, such as the Internet. This can be reduced by providing a set of domain ontologies, but this in itself is a very complex task. Also, it is very difficult to impose a new standard on such heterogeneous structure, and Semantic Web standards provide just one level of standardization in terms of description language and its semantics.

This, we envision the following possible uses of the presented approach on a smaller scale within the Internet:

- Annotating one corporate web site / set of related sites in order to advise user to visit only certain parts of it. For example, a site of non-governmental organization can ask user questions and recommend a series of grant programs he can apply for.

- Combining existing search resources together. Using wrappers for catalogs and keyword search engines we can combine different search facilities to find related pages, even un-annotated. Traditional search techniques here can be used both after reasoning (eg. a set of keywords is auto-generated based on user answers, and then submitted to search engine), before it (initial set of resources is found, and then further processed by expert system) or in any intermediate way.

It has to be noted, however, that if a certain set of ontologies becomes a de facto or W3C standard, it would be much easier to adopt ontology-based search technologies on a wider scale. There are already some attempts to achieve this standardization [Upper Ontology].

4.2. Distance Learning Systems

In modern distance learning systems, one of the problems is automatic course planning based on user's preferences [Panteleev 2002]. Such system, after a consultation with the user or preliminary examination would select a set of materials that has to be studied by the user, and present him with a course plan satisfying his needs in an optimal way. This problem is essentially the same as finding an optimal path in a constrained graph.

To implement such system in the presented framework a domain ontology has to be constructed, and all individual course topics have to be described using dynamic descriptions with constraints. A course plan can then be constructed by the process of distributed inference in the set of resources. Initial state required for this construction can be obtained by questioning the user, possibly including adaptive knowledgebase-driven testing [Malkina 2001].

It has to be noted that a course can be constructed from individual topics that belong to different providers, given that they are described using the same domain ontology. Thus, an issue of standardization is very important topic here, and there are already standards being developed [Panteleev 2002]. However, the ontological approach used in all cases is the same, and thus when certain de facto standards become dominant, it should be possible to adopt proposed technology to use the corresponding ontology.

4.3. Applied Online Intelligent Systems

As an example of using proposed approach for business automation, consider intelligent job search catalog for recruiting agencies. Intelligent recruiting catalog allows companies (employers) to search through the sites of remote recruiting agencies. In this example catalog of recruiting agencies contains knowledge base to define properties of recruiting agencies and some general knowledge about job seekers profiles. All agencies have local knowledge bases to define specific job seekers properties and generate the result list of resumes. In the intelligent selection process, agencies also reference some domain ontologies according to the job in question.

The original idea of the intelligent catalogs to have one primary knowledge base with the general information about the problem area and a set of local knowledge bases for each registered web site. Local knowledge base is used to define specific knowledge for a particular web resource. Predefined top-level ontology is located on the catalog primary resource is available to all parties via inclusion (see Fig.4), for both direct use in reasoning or for introspection. At the same time local knowledge bases may extend the frame structure in order to provide more detailed knowledge. In the architecture ontology (see Fig.3) intelligent recruiting catalog represents a ReasoningCatalog; job agencies web sites are of ReasoningSite type.

Inference starts on the catalog server. Catalog and web sites contain dynamic knowledge in the form of production rules, used by JULIA inference engine running on the catalog site. First step is to define the list of agencies that will take part in distributed inference process [Soshnikov 2002]. An employer will be questioned to define a desired job seeker profile as well as to generate this agencies list, which will help define the initial list of agencies to consult further:

```
ASK agency.location 'Choose the agency location'
      ['Ottawa','Toronto', ... 'Select All'];
ASK job_seeker.profile 'Choose a job seeker profile'
      ['sales','management', ... 'administration'];
```

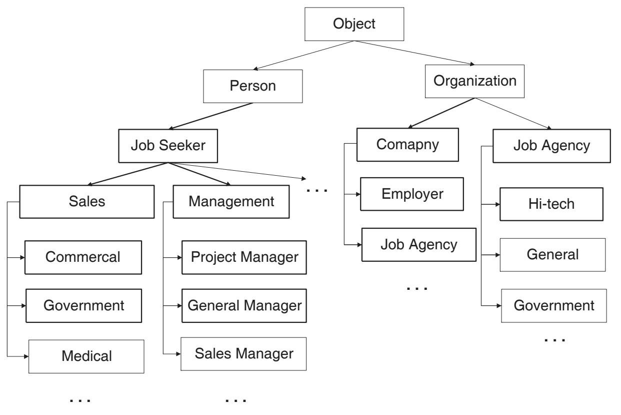

Figure 4: Domain ontology for recruiting portal

When the type of the job is defined, the system looks for all suitable agencies in the specified sector. Since all the government agencies inherit from the `GovernmentAgencies` concept, references to them are automatically made available in `ALL_DESCENDANTS` slot.

```
SET suitable_agencies = FILTER{ %.is_suitable :
     GovernmentAgencies.ALL_DESCENDANTS};
```

The next step is the remote inference using local knowledge bases. The employer will be questioned some specific related questions with the job agency and this particular candidate. A job seeker would basically specify his parameters and the actual frame type, choosing from a range of frames provided by domain ontology.

```
Frame John parent sales_manager {
 Name = 'John Doe',
 Age = 35, Marital_Status = 'single', ... }

FrameSet Cand_Sales parent sales_manager
 SQLRequest SELECT * From Candidates
        Where type='Sales Manager';

SET job_seeker.is_suitable =  job_seeker.salary_ok
    AND job_seeker.qualifications_ok AND ...
SET job_seeker.salary_ok =
  salary_offered > job_seeker.min_salary;
```

The questions will be generated by remote knowledge bases and used by the inference engine on the catalog server. All questions would be presented to the used doing the search through the corresponding interface of Java- or Web-application. As the final result the employer will get the list of references to user profiles that match specified criteria.

5. Conclusions

In this paper, an architecture for multi-agent search system based on ontological descriptions has been presented. Unlike many search systems, it features **online** search strategy, i.e. based on user's preferences formulated either explicitly or in the form of question-answer dialogue, a set of dynamically-allocated resources is traversed, to find those related to user's needs. While this architecture is not well-suited for traditional Internet search on the wide scale because it imposes different standards on resource annotation, it can be used in different projects related to navigation path finding in the set of predefined resources. One example of such approach can be distance learning systems, and sites with intelligent navigation. However, with further development of standard ontologies and description methods, the idea of resource annotation by dynamic knowledgebases can possibly be used for more ambitious tasks.

References

[Semantic Web] Semantic Web at World Wide Web Consortium: `http://www.w3.org/2001/sw/`

[Upper Ontology] Standard Upper Ontology Study Group: `http://suo.ieee.org`

[Soshnikov 2002] D. Soshnikov An Architecture of Distributed Frame Hierarchy for Knowledge Sharing and Reuse in Computer Networks. In *Proc. of 2002 IEEE Int. Conf. on Artificial Intelligence Systems*, IEEE Computer Society Press, 2002. pp. 115-119.

[Sizikov 2002] E. Sizikov, D. Soshnikov Using Dynamic Ontologies based on Production-Frame Knowledge Representation for Intelligent Web Retrieval. In *Proc. of the* 4^{th} *Int. Workshop on Computer Science and Information Technologies*, Patras, Greece, 2002.

[Davison 1998] A. Davison and S.W. Loke LogicWeb: Enhancing the Web with Logic Programming, Journal of Logic Programming, Vol. 36(3), September 1998, pp.195-240.

[Panteleev 2002] M.G. Panteleev, D.V. Puzankov, et al. Intelligent Educational Environments Based on the Semantic Web Technologies. In *Proc. of the 2002 IEEE Int. Conf. on Artificial Intelligence Systems*, IEEE Computer Society Press, 2002. pp. 457–462.

[Malkina 2001] O.I. Malkina, D.V. Soshnikov Creating Adaptive Testing Systems on the Internet using Artificial Intelligence Technologies. In *Sel. Abstracts of* 9^{th} *Int. Student Conf. on New Information Technologies*, MGIEM Publising, 2001. pp.390-392. (In Russian)