The Runner - Recommender system of workout and nutrition for runners

Mihnea Donciu[1], Mădălina Ioniţă[1], Mihai Dascălu[1], Ştefan Trăuşan-Matu[1,2]
[1] Department of Computer Science and Engineering, "Politehnica" University of Bucharest
Bucharest, Romania
[2] Research Institute for Artificial Intelligence of the Romanian Academy
Bucharest, Romania
{mihnea.donciu, madalina.ionita}@cti.pub.ro, {mihai.dascalu,stefan.trausan}@cs.pub.ro

***Abstract*—Recommender systems have been gaining popularity and appreciation over the past few years and they kept growing towards a semantic web. Internet users search for more and more facilities to get information and recommendations based on their preferences, experience and expectations. Nowadays, there are many recommending systems on the web for music, movies, diets, products, etc. Some of them use very efficient recommending techniques (ex. Amazon), while others are very simple, based on algorithms that do not always provide relevant or interesting recommendations. The solution we propose is a recommending system for running professionals and amateurs, which is able to provide information to users regarding the workout and the diet that best suits them, based on their profile information, preferences and declared purpose. The solution mixes a social dimension derived from an expanding community with expert knowledge defined within an ontology. Moreover, our model addresses *adaptability* in terms of personal profile, professional results and unfortunate events that might occur during workouts.**

Runner, nutrition, workout, recommending system, ontology

I. INTRODUCTION

People often look for ways of improving efficiency, ease of use and decrease of time spent for recurrent activities. Additionally, they look for new and exciting things, compare alternatives, are interested in what similar or close people buy or achieve and hope to find a place / system / website that satisfies most of their interconnected interests. To sum up, people want applications that provide them a complete, relevant and efficient solution without having to search for information in other places. Current websites regarding running (for example [8], [9] and [10]) provide partial solutions in the sense that each of them describes certain parts of the process (running techniques, diet, and marathons), but none of them gathers all this information and none of them recommends a solution for a running amateur.

In this domain, most of the existing websites present expert opinions by providing articles from which users only learn general theory. There is no adaptation to users' needs and preferences and systems are inflexible by not being able to give a personalized solution to the user, tailored to his particular interests. On the other hand, websites that give recommendation based on other users' profiles by means of social similarity aren't reliable due to the uncertainty and, sometimes, lack of experience from other members of the community. Users expect from the recommender system to mix these two perspectives (social and expert-based) and to provide them a solution they can confidently rely on. Our unique model addresses both social interaction and knowledge representation, combined with heuristics and evolution modeling, in order to obtain a clear and representative recommendation of diets and of workout exercises.

The criteria of the recommending system that we propose include: user profiles (personal information, hobbies, nutrition and sports preferences), similar users and their recommendations, the evolution in time of each user, accomplishments of workout exercises and diet for a particular day. The system also considers the unforeseen changes possibly inflicted by injuries, illness or other impediments or changes that can affect one's performance.

Currently, the system recommends diet based on the purpose the user defined and considering experts knowledge. This recommending technique is detailed in chapter V.

The knowledge representation model that we introduced in our application is highly suitable as we provide access to resources and the ability to query the data storage and obtain the information in the best fit fashion. The system is based on semantic web facilities and has its information organized in three ontologies. The structure of these ontologies (especially the one for nutrition) was designed and conceived with an expert's help after previously searching for relevant nutritional information on dedicated websites.

These ontologies were populated by crawling the web for relevant data and by parsing HTML pages. We also added some individuals in our semantic repository by developing a distributed web crawler using the Nutch [11] framework. We had to populate the ontologies with thousands of individuals from the web. Since performance of crawling and getting the results was an essential factor, the best solution was to cluster it. We searched the Internet for all the information made public in order to give to the ordinary citizen a hand to become a better runner.

Making a distributed crawler from scratch would have been pointless. The best solution was to integrate existing technologies and extending them with new features that would help distributed crawling for our ontologies. A very good search engine to start from was Nutch, which is based on Apache Lucene [15] and written in Java programming language. It is proper for building ontologies and for clustering

and uses the Map-Reduce mechanism to process this massive data [14]. Map transforms the input into an intermediate representation and Reduce recombines the representation into the final output. That way, we could configure Nutch and develop a plugin in order to get all needed information from the web pages. We prioritized pages related to sports nutrition, sports medicine, training and running facts. This will be further detailed in the following sections.

The following chapter states some facts about our sports medicine research and Section 3 addresses some theoretical concepts about distributed crawling. Section 4 presents the overall architecture, whereas section 5 introduces the knowledge representation model. Section 6 details the actual recommender system, while user opinions and given feedback represent the focus of section 7. The article ends with conclusions and future improvements.

II. NUTRITION

One of the most important aspects of our work is represented by the interdisciplinary knowledge. We aim to build a recommender system that generates the most suitable diet for an every-day runner. This diet has to take into account the needs of a sportsperson and the nutrients that they need to get before, during and after workout. For this purpose, we searched for sports medicine information on the dedicated websites [17, 18] and consulted a nutritionist who gave us the information we needed in order to make our system work as expected.

The way we built our application was based on the fact that, what and when you eat during and after exercise can be just as important. While the pre-exercise meal can ensure that adequate glycogen stores are available for optimal performance (glycogen is the source of energy most often used for exercise), the post-exercise meal is critical to recovery and improves the ability to train consistently. It is also important to consume carbohydrate, such as fruit or juice) within 15 minutes post-exercise to help restore glycogen.

One study found that athletes who refueled with carbohydrate and protein had 100 percent greater muscle glycogen stores than those who only ate carbohydrate. Insulin was also highest in those who consumed a carbohydrate and protein drink.

Consuming protein has other important uses after exercise and we considered introducing daily protein in our recommended diet. Protein provides the amino acids necessary to rebuild muscle tissue that is damaged during intense, prolonged exercise. These facts are very important when one competes in endurance sports, but also when we are dealing with intense effort for a short period of time.

The proper way to accomplish the growth of glycogen ratio is by raising the contribution of carbohydrates. The runner should consume aliments rich in carbohydrates, especially the ones with low glycemic index in the last three or four days, they should try to have a snack at least once at two or three hours. Each meal needs to be centered on potatoes, bread and pasta and therefore, we recommended a daily cereal-based breakfast. The total contribution of energy should stay the same as usual. Smaller portions of aliments enriched in proteins should be consumed, such as meat, fish and eggs. This can be observed from the recommended diet in Figure 3.

After finishing a competition or a workout, the glycogen ratio is very low. We need to consider the speed of the conversion of the carbohydrates in glucose and the muscle carriage. Refilling the glycogen reserves fast is very important for a sportsperson who has more stages in the same competition. Glucose level growth in the blood is monitored by the glycemic index of the aliments and it increases with the glucose level from blood. Studies have shown that consuming high glycemic index carbohydrates and 40g of proteins in the immediate 2 hours after workout, refilling the glycogen reserves is established fast and so, the recovery time decreases.

Runners also get hurt. Even if they take all necessary precautions and avoid dangerous situations, it is still possible for them to sustain a sports medicine injury. Staying on top of common running injury treatments, however, will help them mend faster and get back on the track. Our application gives the user the possibility to report an injury during the workout journal and receive a treatment that would get them back on track in time for the competition.

According to the National Institute of Health [19], the most common sports medicine injuries include sprains and strains, knee injuries, Achilles tendon injuries, and shin pain. If they experience any of these running injuries, one should seek treatment with a physician in the immediate period of time. However, if they are unable or unwilling to see a doctor, the runner can apply the RICE treatment (Rest, Ice, Compression, Elevation) [22] that is the base of our treatment recommendation and is also contained in the developed sports medicine ontology.

III. DISTRIBUTED COMPUTING THEORY

In the following paragraphs, we are going to state some theoretical concepts about the architecture and aspects that helped us develop our distributed crawler.

A. The Map-Reduce model

In the following paragraphs we will describe the map-reduce mechanism, on which our project is based. Map-Reduce [21] is a software framework, introduced by Google, which supports distributed computing on large datasets. The model was inspired by the *map* and *reduce* methods that are often used in functional programming but their purpose has changed significantly.

The model is based on two main steps, map and reduce. The map step consists of reading the input, partitioning it into smaller problems and distributing them to the worker threads. The worker may also distribute the tasks that it receives and also passes the results back to its master. In the second phase, reduce, the master gathers the results from its workers and put them together in order to solve the initial problem and get the final output. The major advantage of the Map-Reduce framework is that it allows distributed processing for both map and reduce operations. The master maps operations to workers that are independent and can be executed in parallel, thus improving time and performance. On the other hand, the number of workers is practically limited by the number of machines from the cluster and the number of cores of each CPU.

The Map and Reduce functions are defined with respect to data structured in pairs of (key, value). Map takes one pair of data with a type in one data domains and returns a list of pairs in a different domain. Applying the Map function for every item from the input dataset in parallel produces a list of (k2, v2) pairs for each call. The framework collects the pairs with the same key from all the lists and groups them together and creates one group for the different generated keys.

The Reduce function is applied in parallel for each group, and produces values in the same domain. Each reduce call produces either an empty return or a new value. The returns of all these calls are collected as the desired result list.

Therefore, we can conclude that the Map-Reduce framework turns a list of (key, value) pairs into a list of values. This behavior differs from the traditional functional programming map and reduce methods, which reads a list of arbitrary values and returns a value combining all the values returned by the map. We found that this model is highly suitable for our project. In this context, we needed to populate the ontology with more than 10.000 individuals, which is a very large number, considering that we have to crawl and parse different data-sources, mainly websites to get the information that we need. The following subchapter will make an analysis of crawling and indexing, as these were used in our project. Each worker node in the Map-Reduce framework engages to return the result and status to its master periodically. If a node fails for longer than an interval, the master considers that node dead and assigns its task to another worker. The reduce operations operate the same way. The master node attempts to reduce operations in the same node, which is highly desirable as it conserves bandwidth across the backbone network of the datacenter.

B. Distributed web crawling and indexing

The first step in our approach involves crawling various data sources, websites mainly, and extracting the data needed for populating our ontology. Web crawling is the process of locating, fetching, and storing the pages on the Web. The computer programs that perform this task are referred to as Web crawlers or spiders. This process has a high importance in fast retrieving the needed information. Also, accessing the information is performed much quicker due to the storage of the pages.

A typical Web crawler „visits" the pages during several steps. First of them would be starting from a set of seed pages. Then it continues locating new pages by parsing the downloaded seed pages and extracting the hyperlinks within. The crawler stores the extracted links in a fetch queue for retrieval and continues downloading until the queue gets empty or a satisfactory number of pages are downloaded.

Web crawling can be performed by choosing one of several crawling architectures. One of them is the sequential model that assumes a single computer and is not scalable. The parallel model consists of multiple computers in a single data center and is not scalable either in terms of network. An alternative for the parallel model is the geographically distributed model, which allows multiple computers in multiple data centers and is scalable, but may have important overheads.

Web crawling performance is described by several quality metrics. One of them is coverage, which represents the percentage of the Web downloaded by the crawler. Freshness is another metric that measures out-datedness of the local copy of a page relative to the page's original copy on the Web.

The page importance is also a significant metric. It consists of the percentage of important or popular pages in the repository. Throughput, scalability in terms of the number of pages, crawlers and data centers, load balancing are also important metrics that we should consider when evaluating the performance of the crawler.

A better alternative to usual web crawling is distributed web crawling that provides a series of benefits in crawling and indexing the web pages. Among these benefits we can state a higher crawling throughput, an improved network politeness due to less overhead on routers and increased availability.

On the other hand, indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. An inverted index is a representation for the document collection over which queries will be evaluated. It is composed of two parts: a set of inverted lists, which contain the document id, word score and word positions and an index into these lists.

Having understood these theoretical aspects, we can state that there are two possible methods for partitioning an index. Term based partitioning consists of distributing T inverted lists across P processors. The second method is document-based partitioning and presumes that N documents are distributed across P processors.

Therefore we integrated the Nutch framework for crawling and indexing in our distributed web crawler project. This process will be described in the following chapters.

IV. OVERALL ARCHITECTURE

"The Runner" is a real-time, web application that was designed on a three-tier architectural model, with bidirectional communication between layers. The general architecture is presented in Figure 1, while each tier will be explained in detail in the following paragraphs.

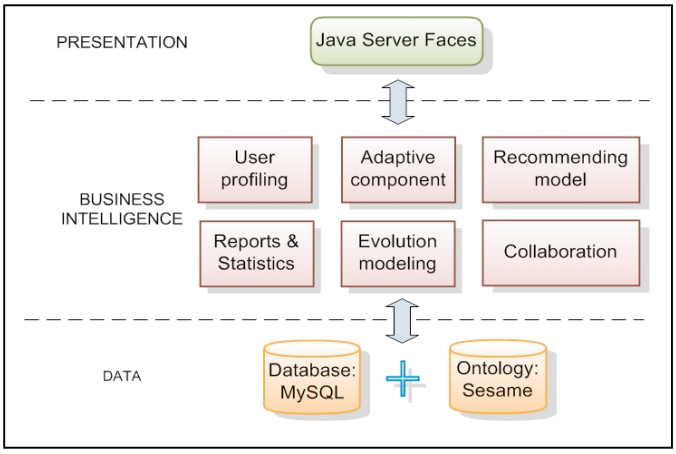

Figure 1. Runner general functional architecture

The presentation tier of our application is represented by a graphics user interface accessible from a web browser and consists of all the controls and graphical elements that help the user interact with the server and access the desired information. We chose to implement the graphical interface using Java Server Faces, because it's an up-to-date, scalable and practical solution for web applications. Java is a object-oriented programming language which proves to be very intuitive and easy to program with. The interface mainly contains textboxes, hyperlinks, images and labels, but we managed to present them in an catchy, simple and usable manner. The user can register, define a profile, a purpose or add relevant information for the nutrition and workout recommendation re-evaluation. All these functionalities are achieved by filling in the text-boxes from the available forms.

The business logic tier is the module in charge of all the queries received from the user, processes corresponding tasks and interacts with the database. This tier is implemented in Java Beans that support the Java Server Faces web pages from the presentation tier.

Business logic is composed of the recommending module ([1, 2]), the adaptive component responsible for personalizing the diet and exercises for each user, the user-profiling module and collaboration modeling between members of the community. This tier is also responsible for providing statistics and reports regarding current performance and for modeling the evolution in time of each user.

Eventually, the database tier is held within a MySQL database management system and encapsulates user profiles, recommended diets and workout exercises. We found this solution suitable for keeping large amount of data, as the number of users will increase in time. Choosing MySQL helped us modularize our data storage and gave us the possibility to keep the important information encrypted.

Knowledge is represented by integrating an OWL [3] global ontology that encapsulates three basic ontologies (nutrition, sports medicine and runners) and includes other existing ontologies on the web. The semantic repository behind uses Sesame [4] as an SPARQL endpoint [5]. As stated in the Introduction section, the nutrition and the sports medicine ontologies were conceived and built taking into consideration experts' knowledge. They are used in recommending diet and treatment for injuries. The Runners ontology will also allow building a social network in our future work.

The OWL file is loaded into Sesame after populating the ontology and then used for extracting the data necessary for the recommender system. We found this process practical because it allows fast connection and the SPARQL endpoint provided by Sesame permits the extraction of information based on filters, which was a clear necessity for our adaptive component.

The model and the implemented architecture represent the best alternative in terms of scalability, modularity required within the Business Intelligence component and performance achieved using only open-source alternatives. Using ontologies allows growth of the amount of information stored and social network users. Also, approaching the application through an object-oriented language as Java eases modularizing.

Figure 2 depicts a page with a proposed workout for a user, in accordance to his purpose. In this case, the purpose represents running 5 km in 25 minutes. The application generated a training program for 2 weeks. For the 31st of August 2011, the proposed workout was to run 1 km in no more than 4min20 sec on the asphalt. The accomplished workout was registered at the end of the day and displayed under the proposed one. As can be observed, the surface type was sand and the time result was worse than expected.

V. KNOWLEDGE REPRESENTATION

One important issue in the implementation of our recommender system was knowledge representation. This kind of application needs an efficient approach considering the fact that the recommendations are made based on specific criteria and near real-time answers are required for user. This process involves a high amount of information that should be easily queried when needed.

In our implementation we chose to represent knowledge using three different ontologies. One of them is ***nutrition-specific*** and contains classes for all types of aliments (vegetal and animal) and nutrients. That way, when the recommended diet is generated, the user will be able to see not only the menu, but its nutritional values.

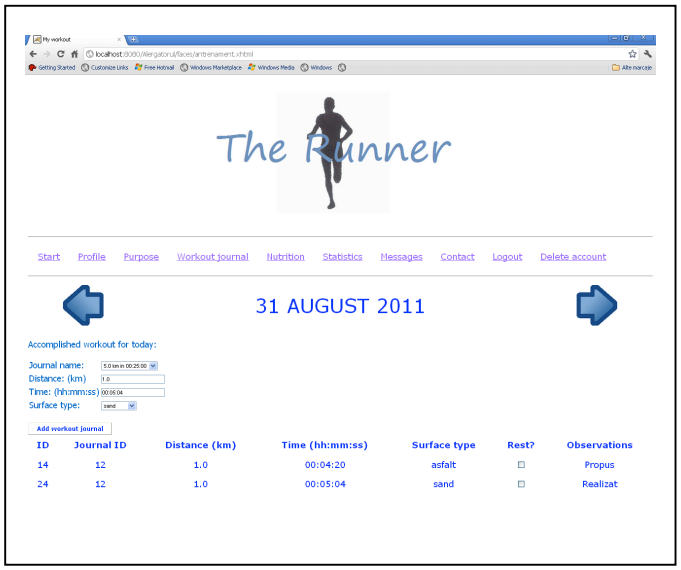

Figure 2. Runner accomplished workout adding page

Another ontology is about ***sports medicine*** and contains classes for injuries – causes, effects and possible treatments. This will be used for recommending a treatment for users when injuries take place. Moreover, we have also built an ontology about ***runners, surfaces and race types***. This will be used for building a social network between the users of the community, in a collaborative manner, around our website. We populated these ontologies by crawling relevant data sources from the web.

The ontology we used the most in our application was the nutrition ontology. For this purpose, we defined classes for the main groups of aliments, dividing them into animal and vegetal aliments. Some of the main concepts of this ontology are:

- AnimalAliment, with subclasses Dairy, Eggs, Fish, Meat (which includes Beef, Chicken, Mutton and Pork);
- VegetalAliment, with subclasses Cereal, Fruit, Seed and Vegetable.

Also, in the same ontology, we have the Nutrient class, with Carbohydrate, Lipid, Mineral, Protein and Vitamin, which are necessary for providing nutritional information about the recommended diets. We designed this ontology so that it contains all the groups of aliments and the runner benefits from all the nutrients, in a diverse manner, with variations in the menu. Having a wide range of concepts in our ontology also enabled us to find more data sources to crawl when populating the domain.

In the sports medicine ontology we included concepts for the important body parts in running, types of exercises, injuries and treatments.

The runners ontology classifies runners and has not yet been populated. It will be populated in our future work with the users of our application, who will form a social network and will represent individuals of a specific type of runner class, depending on their skills and preferences.

The use of the information from ontologies when recommending a solution is made through SPARQL queries, using the Jena API [6]. We also built a semantic repository using the Sesame data storage repository [4] in order to make information more easily accessible.

The ontology-based solution provides efficiency in executing queries and extracting relevant data for our users,

presenting information and more importantly expert knowledge in a semantic web specific manner, tightly connected to Tim Berners Lee concept of linked data [7].

The recommendation of the diet is not yet adapted to the user profile and preferences. This is subject for future work.

When making a recommendation, the system currently takes into account the needs of the human being for all types of nutrients but accents the aliments with carbohydrates for the runners. Therefore, we decided to recommend a daily diet which includes three main meals (breakfast, lunch and dinner) and two snacks.

For breakfast, we chose recommending the users cereals which should be served at 8 am. It's important to start the day with a carbohydrate meal because it provides energy for the whole day, especially when there is an upcoming workout. We give information to the user regarding the glycemic index of these cereals. The first snack is based on eggs and provides proteins for the sportsperson. It's a light, nutritive meal that suits not only the runners, but also regular people. The protein content is also printed when recommending this snack.

When deciding a lunch menu for the runner, we consider the need for lipids and we choose a kind of meat from the ontology (chicken, pork, lamb and beef), giving information about the lipid content and the caloric value.

The second snack is represented by fruits and also provides information about the glycemic index to the user, while the dinner is recommended to be a vegetable-based food, as it brings vitamins and takes into account people's need to eat lighter in the evening.

All of these meals give information about the caloric value and the last column of the recommended diet table (which can be seen in figure 3) shows the total energy value.

The aliment for a meal is chosen randomly from the individuals of the specific concept from the nutrition ontology. Then, the caloric value is computed and the system ensures that there are not more calories than the medium energetic value recommended to a sportsperson (~2200 kilocalories), considering the fact that there is a 200g quantity recommended for every meal.

Moreover, there is a menu recommended for every day of the workout. This period of time is based on the purpose the user defines.

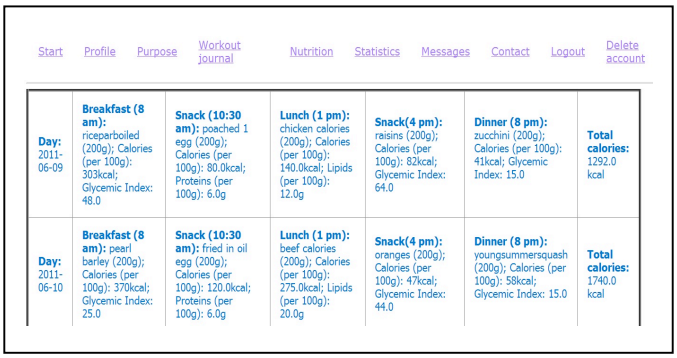

Figure 3. Runner example of recommended diet

VI. CRAWLING AND ONTOLOGY-DEFINED SEARCHING

The need of being always updated with existing knowledge made us integrate a crawling module and text searching engine component in our system. One that best fits our needs is Apache's Nutch [11], a search engine written in Java and based on Lucene, another Apache project. Lucene is a high-performance, open source, full-featured text search engine. Its main features are ranked searching (phrase or keyword searching), flexible (phrases, wildcards, ranges, bools) or field-specific queries (title, author, content) and sorting. Lucene calculates the rank of a document taking into account the uniqueness of each term. Furthermore, Lucene has high scalability and gets fast results. A load balancer is used for distributing HTTP requests to all machines evenly. It is commonly deployed into systems with 10 millions of documents. Nutch has a highly modular architecture, allowing developers to create plugins for media-type parsing, data retrieval, querying and clustering. It divides naturally into two pieces: the crawler and the searcher. The crawler fetches pages and turns them into an inverted index, which the searcher uses to answer users' search queries. The interface between the two pieces is the index, so apart from an agreement about the fields in the index, the two are highly decoupled.

A. *The crawler*

The crawler system is driven by the Nutch crawl tool and a family of related tools to build and maintain several types of data structures, including the web database, a set of segments, and the index. The web database, or WebDB, is a specialized persistent data structure for mirroring the structure and properties of the web graph being crawled. The WebDB is used only by the crawler and does not play any role during searching. The WebDB stores two types of entities: pages and links. A page represents a page on the Web, and is indexed by its URL and the MD5 hash of its contents. Other pertinent information is stored, too, including the number of links in the page (also called outlinks); fetch information (such as when the page is due to be refetched); and the page's score, which is a measure of how important the page is (for example, one measure of importance awards high scores to pages that are linked to from many other pages). A link represents a link from one web page (the source) to another (the target). In the WebDB web graph, the nodes are pages and the edges are links.

A segment is a collection of pages fetched and indexed by the crawler in a single run. The fetch-list for a segment is a list of URLs for the crawler to fetch, and is generated from the WebDB. The fetcher output is the data retrieved from the pages in the fetch-list. The fetcher output for the segment is indexed and the actual index is stored in the segment. Any given segment has a limited lifespan, since it is obsolete as soon as all of its pages have been re-crawled. The default re-fetch interval is 30 days, so it is usually a good idea to delete segments older than this, particularly as they take up so much disk space. Segments are named by the date and time they were created, so it's easy to tell how old they are.

The index is the inverted index of all of the pages the system has retrieved and it is created by merging all of the individual segment indexes. Nutch uses Lucene for its indexing so all of the Lucene tools and APIs are available for interacting with the generated index. Since this has the potential to cause confusion, it is worth mentioning that the Lucene index format has a concept of segments, too, and these are different from Nutch segments. A Lucene segment is a portion of a Lucene index, whereas a Nutch segment is a fetched and indexed portion of the WebDB [12].

Crawling is a cyclical process: the crawler generates a set of fetch-lists from the WebDB, a set of fetchers downloads the content from the Web, the crawler updates the WebDB with

new links that were found, and then the crawler generates a new set of fetch-lists (for links that haven't been fetched for a given period, including the new links found in the previous cycle) and the cycle repeats. This cycle is often referred to as the generate/fetch/update cycle, and runs periodically as long as you want to keep your search index up to date.

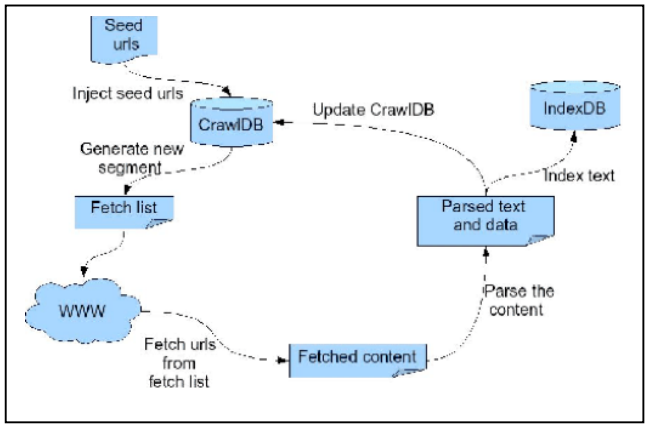

Figure 4. The crawl graph

URLs with the same host are always assigned to the same fetchlist. This is done for reasons of politeness, so that a web site is not overloaded with requests from multiple fetchers in rapid succession. Nutch observes the Robots Exclusion Protocol, which allows site owners to control which parts of their site may be crawled. The crawl tool is actually a front end to other, lower-level tools, so it is possible to get the same results by running the lower-level tools in a particular sequence.

After creating a new WebDB, the generate/fetch/update cycle is bootstrapped by populating the WebDB with some seed URLs. When this cycle has finished, the crawler goes on to create an index from all of the segments. Each segment is indexed independently, before duplicate pages (that is, pages at different URLs with the same content) are removed. Finally, the individual indexes are combined into a single index [12]. Figure 4 shows a simplified version of the entire crawling process.

B. The searcher

The Nutch search system uses the index and segments generated during the crawling process to answer users' search queries. Nutch looks for results in the index and segments subdirectories of the defined directory.

One of Nutch's key selling points is its transparency. Its ranking algorithms are open source. Nutch's ability to "explain" its rankings online (with the explain link in the web interface) takes this one step further and allows an (expert) user to see why one particular hit ranked above another for a given search.

The central search class is NutchBean. Upon construction, the NutchBean object opens the index it is searching against in read-only mode, and reads the set of segment names and filesystem locations into memory. The index and segments locations are configured in the same way as they were for the web application: via the searcher.dir property. Suppose we want to search the keywords of the class FattyAcid of our nutrition ontology: „fatty acid fat lipid". The Query.parse() method invokes Nutch's specialized parser (org.apache.nutch.analysis.NutchAnalysis), which is generated from a grammar using the JavaCC parser generator. With a Query object in hand, the system can now ask the bean to do the search for us. Finally, a NutchHits object is returned, which represents the top matches for the query. This object only contains the index and the document identifiers. To return useful information about each hit, we go back to the bean to get a HitDetails object for each hit we are interested in, which contains the data from the index.

The last piece of information that is displayed by the application is a short HTML summary that shows the context of the query terms in each matching document. The summary is constructed by the bean's getSummary() method. The HitDetails argument is used to find the segment and document number for retrieving the document's parsed text, which is then processed to find the first occurrence of any of the terms in the Query argument [13].

Even if it is a Java project, Nutch uses Hadoop (another Apache project) that actually invokes various GNU utilities to perform some operations, usually on the file system (bash, chmod, df, ls, tail etc.). This is the reason why making it work in Windows requires the use of Cygwin. Our solution is a Unix like operating system and our choice was Oracle Solaris 10.

VII. THE RECOMMENDING SYSTEM

The recommending system allows us to follow accurately user's needs and purposes in every moment ([1, 2]). In this way, adaptability plays a central role, adjusting diet and exercise requirements for every individual.

Firstly, it is most useful to find similarities between different user profiles. One can see how many people are in a similar position and even the methods they used to achieve their personal goals. When it will be available, this feature will help the evolution of their capabilities and will encourage them to increase their performance, surpassing others in a competitive and sporty manner.

When the user is new and there are no relevant similarities for him, the classic problem of a "cold start" arises. This is already implemented using heuristics in order to match his goals and defined intentions with a predefined pattern. For example, our database contains a training program for running a marathon on flat surface in less than 4 hours and a user wants to run it in three and a half hours. Additionally, training must take place in two months maximum when a normal workout would last three months. The heuristic computes the best fit for these constraints and already creates an approximate training program. Results are later on adjusted based on the actual user feedback during and at the end of the program.

Predefined training programs are added to the database via the Administrator interface. The system administrator can add generic purposes to the database, specifying the distance, the target time, the surface and the number of weeks required for training. After that, he must fill the form for each training week with the workout for every day. The generic training program will be stored into the database and will be the starting point for all automatically generated training programs.

Besides the workout setup, our system will be able to addresses social diet and nutrition recommendations. Excluding the "cold start" situation of a newly entered user or exercise program, the system proposes recommendations using the Jaccard coefficient and the cosine similarity

An addition to our recommending system will be the possibility of defining constraints for each user profile, similar to the classic Constraint Satisfaction Problem. In the case a user doesn't want to include milk or pasta in his diet or maybe he enjoys better other nutrients, our system shall take into

account his desires. Our knowledge representation of ingredients and their caloric index is useful for finding the best proportions of meals. If at some point the user changes his desires and replaces some constraints, the recommendation will change accordingly. This will avoid monotony for diets because some nutrients can be easily replaced by others, meanwhile ensuring the same effect.

One of the novelties of our application is that it brings the possibility to recommend a special diet and workout especially for runners. So far there are no recommender systems suitable for runners. There are systems that recommend a diet for overweighted people who just want to loose weight. These systems give the user the possibility to fill a profile with information such as height, weight, age, gender and then generate a hypo-caloric diet which helps them loose weight.

Our application manages to mix all categories of nutrients, precisely focusing on carbohydrates, which are the most important for providing energy to a sportsperson. In addition, not only the system generates a suitable diet, but adapts this diet to a corresponding workout journal which considers a determined period of time which the runner wants to succeed within, the surface type and the time the runner needs to accomplish for the competition. Also, the novelty includes the facility to give the user the possibility to add an injury during the workout period and the system regenerating a whole new journal based on their medical condition.

Moreover, The Runner system represents a mixture between experts' knowledge and a social dimension in generating the nutrition and workout journal. It combines the medical theory with the user's needs, considering their profile, their nutritional preferences, their purpose and the similarity to their friends from the social network.

Having stated the above, our web application ensures novelty among the existing applications, whereas future improvements consider the integration with other applications.

VIII. Implementation

The implementation of this project is divided in two separate streams. The first addresses the distributed web crawling, using the Nutch search engine, while the second is centered on the recommending system.

Crawling is the first part of the entire process. First of all, the URLs directory will contain one or more files with links of the pages to be crawled. Since it is an iterative process, we created a script that uses two parameters – iteration size and depth. The first one decides the number of given web links to be fetched and the number of threads that will run at every step. The other one determines the depth of the generate/ fetch/ update cycle for every step. That is, how deep will the parser go when fetching the links found in the already parsed web pages. The script injects the links from the URLs directory into the crawl database. Then it generates the fetch segments, fetches them and updates the crawl database with the new found links from the corresponding pages. Finally, the links are inverted to make them match their anchors and all the fetched pages are indexed into the local index database. If there are no reported errors or exceptions during the automated crawling process, the just created local database is ready to be searched.

The next step is to search the database with keywords from our ontologies. After creating an ontology model using the Jena framework API, the Nutch OwlParser is used to parse the root classes of the model and then all the subclasses. For every class, the search method extracts the class local name and the description, which contains the keywords proper for the search. All the results are printed in an output file, with title, URL, date fetched, summary and the entire parsed text in which they appear. The recommending system is also written in the Java programming language and the user interface is built using the Java Server Faces framework.

When the user connects, a new session is created as a Session Bean. An entry with his username is added in the session table of the database. The user can then choose to see its own profile, the calendar with workouts, he can set a new running purpose to achieve in the future or see some useful information about nutrition.

When adding a new purpose, the required variables – distance, time, proposed date and surface type – are registered as a new entry in the Purposes table of the database. Then, a new journal entry is added in the journals table with the name of the purpose, expressed as "<distance> km in <time>". The next step is the calculation of the difference of days and weeks between the current date and the date proposed for achieving the purpose. The trainings table is then queried to find the list of trainings that best fit to the specified variables of the purpose. The selection criteria are the following:

a) Smaller difference between the existing training distance in the database entry and the proposed distance to run

b) Smaller difference between the time to run the proposed distance and the time of the existing training in the database

From this list, the best fits are chosen by trying to minimize the difference of the weeks between the required weeks and the remaining weeks until the proposed achievement. It may be possible to have more than one selected training program at this point. The next criterion is to match the required running surface. If not included, then the first program is chosen for the runner.

At this point, all the daily workouts and nutrition recommendations for the selected program are extracted from the Generic Workouts table of the database, limiting them to the minimum number between the days left and the required days for this training program. Of course, if there are not enough training days left, the final results for the runner may be unsatisfying. If there are more days left than required, then the training program will be more relaxed and will have more days for resting between workouts. Additionally, the user can determine glycemic indexes, nutritional values, proteins or low fat meat. All the data were extracted after crawling the web with the Nutch searching engine and populating the nutrition OWL file. After creating an ontology model from the ontology file, proper SPARQL queries are executed, according to the user's choice of view. The results are all returned as of List object types and rendered in the JSF XHTML web pages.

Other available options are: viewing some statistics about the completed and remaining workouts and sending messages to the Administrator or other users registered in the database.

IX. User Opinions And Recommendation Relevance

Real user experience and opinions are the best benchmarks for a recommending system. After receiving feedback from about 5 different persons, including a nutritionist doctor, we came to the conclusion that the variety and personalization of results is the key to success. A questionnaire was presented to the testing candidates for evaluating some relevant criteria like

usability of the application, ergonomics, relevance of the results, esthetics and interaction ability. Having an expert opinion in this case is very helpful, especially in recommending diets, because this is the best way of addressing needs and desires of people who go out running.

Users were delighted by the simple and fine esthetics of the site. Most of all, our Runner logo with a person who looks like it's running out of the screen was very inspired. It gives each user the feeling that they could be that runner, no matter their gender or age. Giving a sense of trust is a psychological key for keeping the registered users active in our system and to attract others to join our community. Most users saw this application as their first step towards performance. For emphasizing this aspect, our intention is to give a hand to beginners and to keep the performers in shape. Top-level athletes are not our target runners, but they could maybe take into account our recommendations, too.

A useful lesson learned from a user who used two different training programs for running marathons at different times was that making less than 100% of the workouts was not a problem for him to achieve his personal final goal. He used the 4 hours marathon (42,195 km) training program, but completed around 70% of the workouts. The results were better than expected: he finished the marathon in 3h58min and was very happy about his accomplishment. In conclusion, having this recommending system which guides you to accomplish the most important workouts like long runs and repeated sprints, in despite of resting when a medium run is programmed, can help you achieve your final purpose.

X. Conclusions And Future Improvements

This adaptive recommending system, designed for runners who want to achieve their goals and performances, combines expert knowledge with the social dimension. Dynamic adaptation to social changes makes our approach up to date and relevant.

Our approach towards the Social Semantic Web is enforced by extracting all required information directly from the web and by searching the keywords in our ontologies stored as semantic repositories. This data is built from multiple users' activities, with all the profile characteristics, purposes, trainings and recommended diets stored in our database. User interaction is currently possible by exchanging short messages between each other and will be more interactive when our platform will transcend into a true social network. Users will have the possibility to watch other profiles, see workout exercises and publicly available programs, make physical improvements, score better running results and finally they will have all the required means of achieving their purposes. In addition, it will be more easy for them to make an appointment to go for a short evening run in the park or to establish a meeting place before the beginning of a contest.

In the future, the possibility of continually tracking a runner using a GPS device will be integrated for recording precisely the route, with its characteristics. From this point, Google maps will provide a visual interface for viewing the track. Another direction of future improvements is to develop a small size application for Android smartphones that would make the process even more dynamic.

The ontology that we used for extracting our data in the recommending system will be improved by crawling more data sources, using more machines to split the URLs and, therefore, populate the ontology with hundreds of thousands of individuals, which will significantly improve accuracy and diversity of the recommended menu and treatments.

References

[1] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*. ISBN 0471228524, John Wiley & Sons Ed., 2003.

[2] Olivia Parr Rud. *Data Mining Cookbook*. *Modeling Data for Marketing, Risk, and Customer Relationship Management*. Wiley Computer Publishing, 2001.

[3] OWL 2 Web Ontology Language Document Overview, W3C Recommendation 27 October 2009: http://www.w3.org/TR/owl2-overview/ [May 5, 2011]

[4] Sesame: RDF Schema Querying and Storage: www.openrdf.org/ [May 5, 2011]

[5] SPARQL Working Group Wiki: http://www.w3.org/2009/sparql/wiki/Main_Page [May 5, 2011]

[6] Jena – A Semantic Web Framework for Java: http://jena.sourceforge.net/ [May 5, 2011]

[7] Tim Berners-Lee. *Linked Data*. International Journal on Semantic Web and Information Systems, volume 4, issue 2, publisher W3C, ISSN: 15526283, 2006.

[8] Sanathon: www.sanathon.ro [May 5, 2011]

[9] Ro Club Maraton: www.maraton.info.ro [May 5, 2011]

[10] Runners Club: www.runnersclub.ro [May 5, 2011]

[11] Nutch: http://nutch.apache.org/ [May 5, 2011]

[12] Tom White. *Introduction to Nutch, Part 1: Crawling,* Java Today: http://today.java.net/pub/a/today/2006/01/10/introduction-to-nutch-1.html [May 5, 2011]

[13] Tom White. *Introduction to Nutch, Part 2: Searching*, Java Today: http://today.java.net/pub/a/today/2006/02/16/introduction-to-nutch-2.html [May 5, 2011]

[14] Ashish Thusoo, Joydeep Sen Sarma et all. Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, Volume 2 Issue 2, 2009.

[15] Apache Lucene: http://lucene.apache.org/ [May 5, 2011]

[16] Dhruba Borthakur. *The Hadoop Distributed File System: Architecture and Design*: http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf [May 5, 2011]

[17] SportsMedicine: http://sportsmedicine.about.com/cs/nutrition/a/aa0814 03.htm [May 5, 2011]

[18] Top Sanatate: http://www.topsanatate.ro/articol/nutritia-in-timpul-antrenamentului-sportiv-19367.html [May 5, 2011]

[19] National Institute of Health: http://www.nlm.nih.gov/medlineplus/sportsinjuries.html [May 5, 2011]

[20] Orthopedics: http://orthopedics.about.com/cs/sportsmedicine/a/runninginjury.htm [May 5, 2011]

[21] Map-Reduce http://labs.google.com/papers/mapreduce.html [May 5, 2011]

[22] RICE: Rest, Ice, Compression, and Elevation for Injuries, University of Michigan Health System website, http://www.uofmhealth.org/health-library/tw4354spec #tp21229 [July 28, 2008]