An Architectural Solution of Assistance e-Services for Diabetes Diet

Vasile AVRAM[1], Diana AVRAM[2]
[1]Academy of Economic Studies, Bucharest, Romania
[2]SIGMA Publishing Co., Romania
vasileavram@ie.ase.ro, diana@avrams.ro

The aim of this paper is to outline the requirements and main architecture for a useful tool for determining the nutrition facts of food for people having Type 2 Diabetes. This diabetes is used only to establish the target audience, a "mass of people" having, maybe, to less in common regarding the computer usage skills. The characteristics of the target audience (huge number, diversity of habits and behaviors, computer usage skills) requires a solution based on web services delivered at least partly as a standalone/ portable application, build from Web services and provided with means for domain knowledge dissemination and usage.
***Keywords**: Software Architecture, Knowledge Management, SIK, Business Rules, Type 2 Diabetes*

1 The Problem Space

The problem here starts from the fact the diabetes once diagnosed is for life. It occurs either because of lack of insulin, called Type 1 Diabetes or insulin dependent, or because of presence of factors that oppose the action of insulin (this is due either to diminished insulin secretion or in increased hepatic glucose output) and called Type 2 Diabetes. Type 2 Diabetes results from the body's ineffective use of insulin and is very common in most developed and developing countries. According to Watkins [14] the evolution of number of people aged more than 20 years estimated to have Type 2 Diabetes is:

- In developed countries: 1995-48 Millions, 2000-52 Millions, and expected in 2025-70 Millions;
- In developing countries: 1995-85 Millions, 2000-100 Millions, and expected in 2025-230 Millions.

According to World Health Organization (WHO) more than 220 million people have diabetes [15]. The common consequences of diabetes represented by that it increases the risk of heart disease and stroke, foot ulcer, retinopathy with direct impact on blindness, kidney failure, damage to nerves etc. According to WHO [15] diabetes and its complications have a significant economic impact to individuals, families, health systems and countries. WHO [15] estimated that China, for example, will lose about $558 billion in foregone national income between 2006 and 2015.

Prevention of Type 2 Diabetes can be easily realized by changing the lifestyle by weight reduction, improved diet (less fat, less saturated fat, carbohydrates control, and more dietary fiber) and increased physical activity [14].

The aim of this paper is to outline the requirements and main architecture for a useful tool for determining the nutrition facts of food for people having Type 2 Diabetes. The solution is intended to have enough generality to cover most problems related to diet and content of foods in different nutrients.

This Type 2 Diabetes is used here only for the scope to establish the target audience of an information system helping those people in preventing and/or reducing the effects by an adequate alimentary diet and for assisting them in determining the proportion of different nutrition facts in their food and combination of foods.

The general scope of the information system is:

- To supply to his users access to technical domain specific knowledge (conversions between different scales of measurement, equivalences, etc);
- To supply food preparation recipes specific to some diet;

- To be a universal repository of the domain knowledge by collecting existing and new defined recipes/ cooking or preparation procedures;
- To assist users in determining the characteristic indicators (nutrition facts) when cooking food, in establishing a specific diet etc;
- To ensure the users privacy;
- To ensure the property rights of the authors of recipes/ procedures.

According with this scope the target audience is characterized by:

- Is globally located does no matter the economical development stage of the country they belong to;
- It includes people having different cultural behaviors, habits, and even ages (under last facts, from children's to adults);
- It includes people from diverse social strata, with different levels of education and various skills and knowledge in using computers (on a "dumb" to "expert" scale, we can say);
- It not excludes the adherence to the target audience of business users wanting to use the knowledge managed here to operate their business or parts of that.

All these characteristics of the target audience will have a strong impact to the informatics solution at least from the point of view of human computer interface, personalization and deployment, maintenance and operation, as outlined in what follows.

The services required by the target audience represented at least by the following:

- Documenting in this domain;
- Searching for recipes specific to a diet;
- Determining nutrition facts of the food they cook;
- Defining new recipes and/or cooking procedures etc.

2 Software Architecture

We consider here the following common definitions, from the indicated sources, for software architecture:

- 'Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution' [2];
- 'Software architecture presents a view of a software system as components and connectors. Components encapsulate some coherent set of functionality. Connectors realize the runtime interaction between components' [1];
- 'The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them'[7].

By analyzing these definitions we can argue that the software architecture defines structure of an application as a set of inter-related components that meet specific requirements and constraints.

A key structural element in the software architecture is represented by the minimization of dependences between components to obtain a loose coupling architecture and by eliminating the intermediary components and their required dependences, ensuring in that way the agility, the adaptability of the solution to a large scale requirements and changes.

This can be realized using a service-oriented approach in system design, realization, and implementation. The main goal of service-oriented approach is to change the user centric web into a service (application) web centric, it means to make web applications cooperate and understand one another. We consider here that 'a service is a self-contained software element that provides well-defined business functionality and an interface that is abstracted out and separate from the implementation of functionality [14].

A service-oriented architecture is a software structure, for building applications that implements business processes or services, using a set of black-box type software components loosely coupled and orchestrated so that they deliver a well defined level of

services [13]. In that way the design of the application follows the trend in application design by the fact it replaces the rigid structures (and rigid architecture) with a flexible architecture allowing application integration.

By organizing the software system around the central concept of business process and the business process drives the behavior of the components in the system we obtain a process-centric architecture as defined in [14].

In the context of this paper the business processes are executable in computer as code. Since the software here is all about the system concerned we can argue that the system is a software intensive system it means "a system where software contributes essential influences to design, construction, deployment, and evolution of the system as a whole" [2].

3 Web Services and Application Architecture

A Web service forms a distributed environment of objects, called services, in which each such object performs a specific task or a set of tasks and can be accessed remotely via standard interfaces. The architecture of Web services is based on principles and standards for connection, communication, description, and discovery. Web service uses a three-tiered model (Figure 1), defining three actors: service provider, service consumer, and service broker. The main role of these actors is:

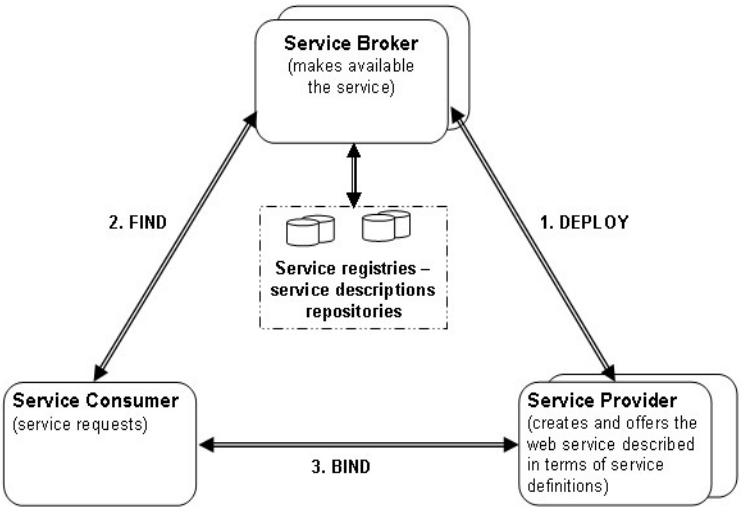

Fig. 1. Web services model

- *Service provider*: creates, defines and deploy the Web Service. By deploying the Web service it advertise the service to potential users, humans or agents. A service provider creates a Web service (the implementation of) and its service definition and then publishes (1) the service in a service registry (or directory) based on the standard Universal Description, Discovery, and Integration (UDDI) specification. The description of the service must be readable for both, humans and machines, as potential users. The service repositories have the role to collect and organize location and description information and make it available to any clients that need it. The service repository provides the mechanisms to facilitate the publishing of the services by service providers and to enable service clients to locate the service and the associated binding information. The Web service publishing is the operation to record or advertize into the registry and acts as a contract between *Service repository* and *Service provider*;
- *Service broker*: maintain the service repository of published services and searches for services that best meet the user requirements and deliver this information to the user. Here the role of *Service repository* is simple: it must signal

the match between the user request (*Service requestor*) and *Service provider* (supplier) description. Once found a match is established a direct relationship between the *Service requestor* and *Service provider*;

- *Service consumer (requestor)*: is a client (a human or a software agent) that intent to use a service for a specific goal and that addresses a request for that. Once a Web service is published, a service requestor may find(2) the service via the UDDI interface. The UDDI registry provides the service requester with a WSDL (Web Services Description Language) service description and a URI (Uniform Resource Identifier) pointing to the service itself. The service requestor may then use this information to directly bind (3) to the service and invoke it. Bind creates a client-server link between *Service requestor* and *Service provider*.

For both service consumer and service broker the Web serviceis available as an interface that describes a collection of operations that are network-accessible through standardized XML messaging. If a service provider goes down, the broker can always direct consumers to another one. There are many brokers so consumers can always find an available one.

3.1 The target audience

Broadly, the target audience, the users, can be categorized in two categories:

- *Business users*, such as restaurants for example, searching for a specific dietary recipe or wanting to determine the nutrition facts for a food they cook, that uses the services as defined by standards on a subscription base. This category include in fact all users wanting access to the protected information, the recipes, working procedures, and generally knowledge and information subject to copyright law and for which the owners want earning revenues;
- *Individuals users*, the "mass of people" formed by individuals that requires the services without any charge, for free.

In what fallows we focus more on that last user category, individual users. In the case of individuals users, having a "mass of people" as target audience the application functionality, even defined as a collection of Web services, must be accessible from a general Web browser in an Internet "natural way", as human meaningful and accessible as possible. It implies also to make the data storages and knowledge repository an integral part of the rest of functionality of the system.

The environment is heterogeneous, disparate and multiple. The system must effectively adapt its behavior according to changes in the business process that generally triggered by changes in business requirements. This requirement creates the premises for new design approaches of Web services and their functionality: the possibility to define even the human-computer interface (HCI) in terms of Web services using other Web services.

3.2 The application architecture

Because the application is interactive this must ensure for a fast and consistent response to interaction. When a remote service is invoked, the speed at which the response is generated is determined not just by the load and performance of the server and the network but also delays in all the software components involved – the client and server operating systems' communication and middleware services (remote invocation support, for example) as well as the code of the process that implements the service. Thus another requirement generated by performance reasons such as throughput, response time, and deadlines is to transfer in same way almost processing and storage to the client computers (workstations) for all individual users. This is imperatively required at least by the fact the services intend be free of charge for the individuals users category.

The question here is in fact a request to change the approach and delivery of web services for that category. If a service can be replaced with another having the same functionality (the way realized and

implemented differs maybe), selected and offered by a broker, a human-computer interface service should be kept as such until the user quit it, in a similar way the user keep a traditional application.

Note that the solution just outlined is only for the individual users category and do not apply to business users for which the delivery of services is realized naturally as specified by the standards governing the domain of Web services. The services available for the business user category, the paid ones, are available via a multitude of brokers and implementations.

The user interface, based on dialog windows and/or printed reports, should take in account elements such as size, scrolling, navigation, partitioning, information hiding, highlighting and printing.

This must be build with respect for general requirements for designing the user interface of application programs such as:

- The user must be aware always about what must be done in the next step;
- The consistent structuring of the screens (messages, instructions, commands or information should appear in the same general shape or format);
- The usage of default values for the input fields (determined as a majority average for usage of the cases by the target audience);
- The user must be able to choose between a menu-driven, as the oldest and most commonly employed strategy, and an instruction-driven interface, with a dialog based on an instructional set (a command language interface) using a natural language like syntax.

The user interface must act as an initiator of the functional services composition (build).

The data stored must be accurate to the business process and the changes in the system must be realized with a minimal effort. The concrete web technologies used and technical solutions should facilitate the installation and deployment, without or extremely minimal effort from the part of user, and must be platform independent.

When keeping and exploiting both data and computation in a "centralized" like manner (even both replicated and shared) the balancing of computational loads is difficult be maintained at a satisfactory and efficient state. The solution do not exclude the existence of a "traffic-like police officer", a server dedicated to control the free of charge access flow to services, that must be accomplished in a satisfactory manner, from paid access flow to services that must be satisfied with respect of some high quality criteria.

Figure 2 introduces the application architecture as for the client workstation and for the server.

3.2.1 Client side components

The components at client side (user workstation) are briefly described in the following paragraphs.

Data & Knowledge Collection. Comprise a number of loosely coupled processes that:

- Explicitly collect user preferences and requirement;
- Transparently track the user relevant activities and store them, depending on the case, in the *Data Store* or in the *Knowledge Repository*. In this information system the users makes up the vast majority of the information workers. Since they concerned with the functionality the system provides, with the system's ease of learning, and ease of use their interactions and habits with the system will be monitored and collected as status information used for future analysis and system changes so that this in turn be able to adapt even to particular situations and not only to common uses. The captured events relates to user preferences, consulted recipes, and new defined recipes.

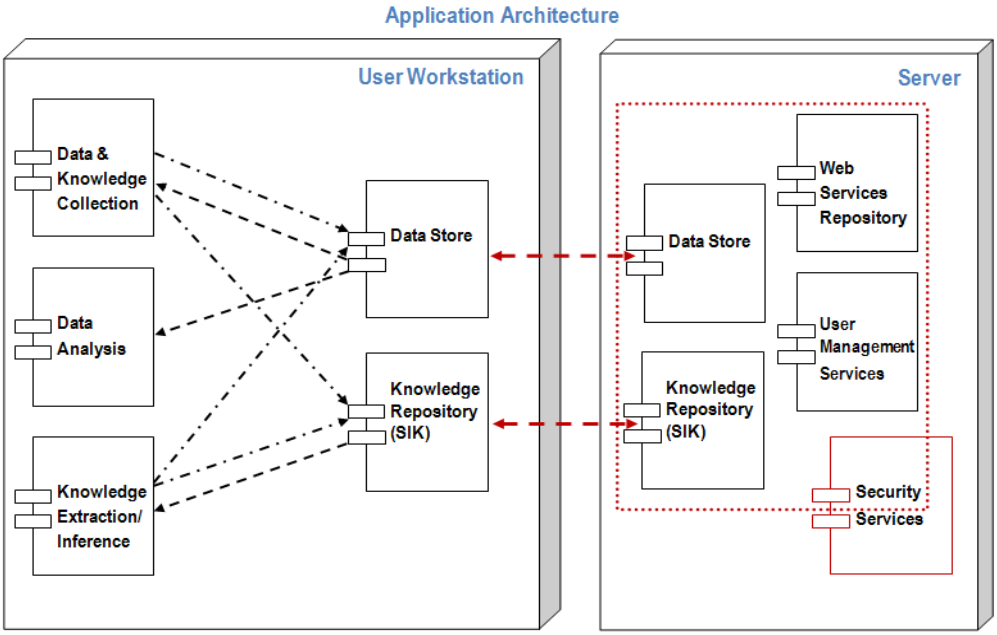

Fig. 2. The application architecture

Data Store. This component comprises a relational database used to store nutrition facts about basic products, used to prepare the foods. It contains also large objects such as products images, product categories and subcategories images allowing define a real nutritional catalog of products. It comprises also a set of tables, only in the locally database structure, to store and maintain the user preferences. The server side collects all products and nutrition facts added by users. They require a human validation before be included permanently into the server *Data Store* and made available for all users. The validation refers to check the correctness of nutrition facts only for the base products since for recipes this is automatically determined from the constituents. At server side, a set of tables, are used to collect statistics about user preferences. The main structure of the data store includes the tables and relationships used to maintain nutrition facts as illustrated in Figure 3. This structure is an excerpt from the Diet database structure used for both Microsoft SQL Server and My SQL implementations.

What is characteristic to this structure is that the Products table maintains information about the four main products nutrition facts energy value, proteins, lipids, and carbohydrates necessary for survive, given as content per 100 grams from product. Generally these nutrition facts provided as information for consumer as a label on the packing of food. The products can have also other nutrition facts needed for health in different dietary regimes such as fiber content, (un)saturated fat, gluten etc. All this extra nutrition facts a product may have are signaled by a flag like attribute, the attribute "*extraattribute*" of the *Products* table, and collected in a separate table called "*Extraattributes*" where is specified his percentage content. The main structure maintains the information required for a nutritional catalog of products. Products can be identified by the "EAN 13" international code used for products. These data will be used in the process of creating new recipes definitions to be stored as knowledge in the knowledge repository.

A design alternative for that component is to be realized as an XML representation of the relational database as described in Bos [10] and the XML query language XQuery for data manipulation or by exploiting the XML DOM model and using scripting languages.

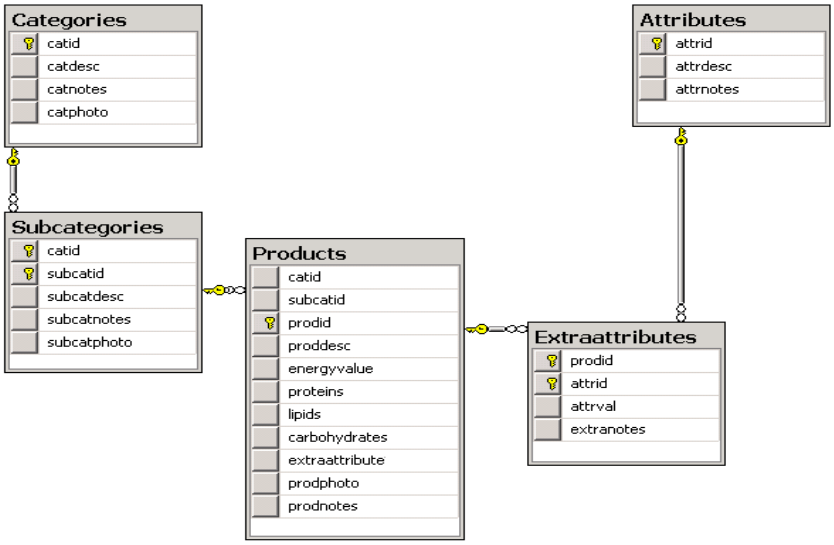

Fig. 3. The tables and relationships used to maintain nutrition facts (Diet database, partly)

The XML DTD (Document Template Definition) equivalent to this schema can be found at http://www.avrams.ro/diet.dtd together with a database file structured accordingly with http://www.avrams.ro/dietdb-02.xmland an example of manipulation of the stored data by intermediate of scripting languages http://www.avrams.ro/showproducts.html.

Figure 4 shows some excerpts from these files with the scope to illustrate this portable and integrative solution. Every time an XML document referencing XML DTD documents this is checked for compliance with the definitions given in the document template. When using the XML DOM, to access the records, the database is loaded/ parsed as a tree whose elements can be processed by DOM functions and commands. The right side of the figure contains an excerpt from an HTML page that displays, by intermediate of a JavaScript, the products from catalog as illustrated in the rightmost down box.

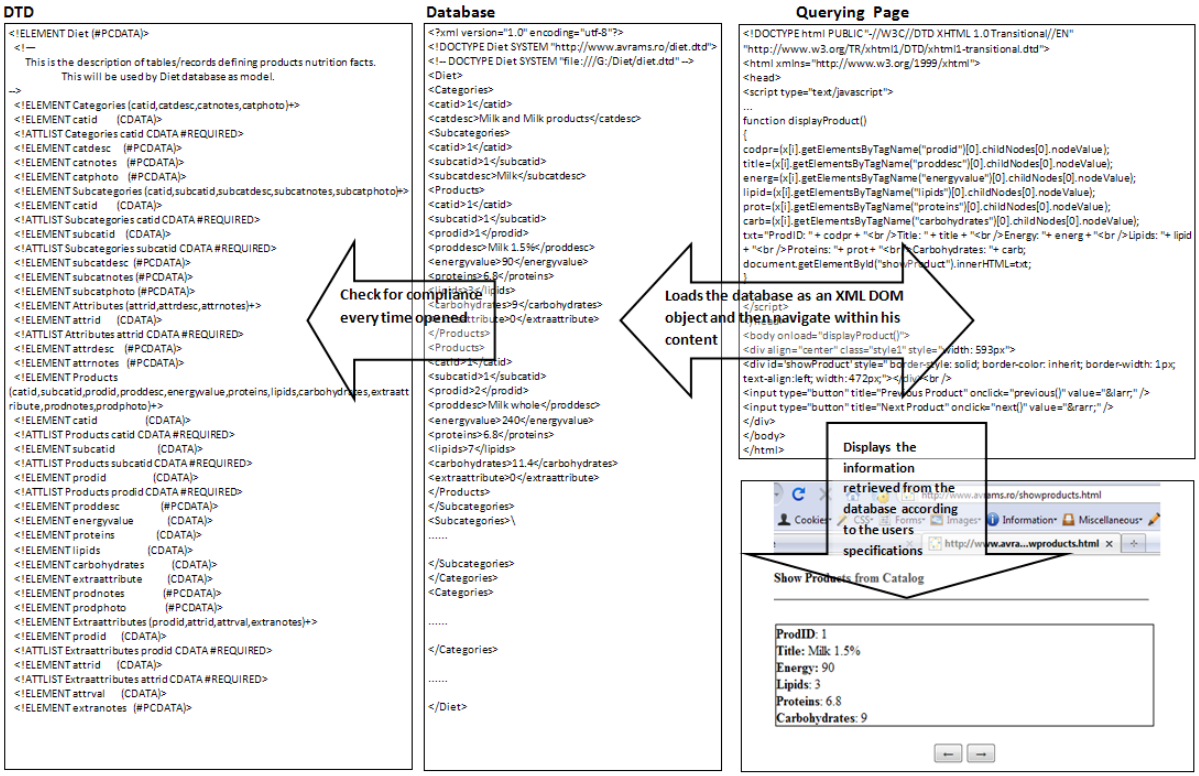

Fig. 4. The model used to query an XML database by using XML DOM

Knowledge Repository. It contains recipes definitions, knowledge about diabetes, nutrition facts, conversion procedures and formulas, equivalences between different measurement systems etc., or in other words the business rules of the domain. The knowledge repository is interfaced with end user services for creating new recipes definitions and user services to consult the knowledge. The creation of new recipes takes place by consulting the *Data Store* and then building the associated RDF (Resource Description Framework) definition to be stored as new knowledge. Once created a new recipe this one do not requires, when consulted, to access in turn the constituents in the *Data Store*. The *Data Store* will be consulted only in maintenance operations and/or to check for consistency of definitions. It contains also all domain knowledge incorporated in the Web services that contributes to the diet application functionality as a cumulative SIK (Figure 5) as defined in Avram [3] [4], and used for extensive knowledge management. SIK is used for systematically acquire, structure, store and maintain knowledge, formalized as business rules for all domain business rules that are incorporated in the software product (web services or the application build on their basis) itself together with the specific knowledge of the domain [5]. It can include also knowledge about the operation of the current application.

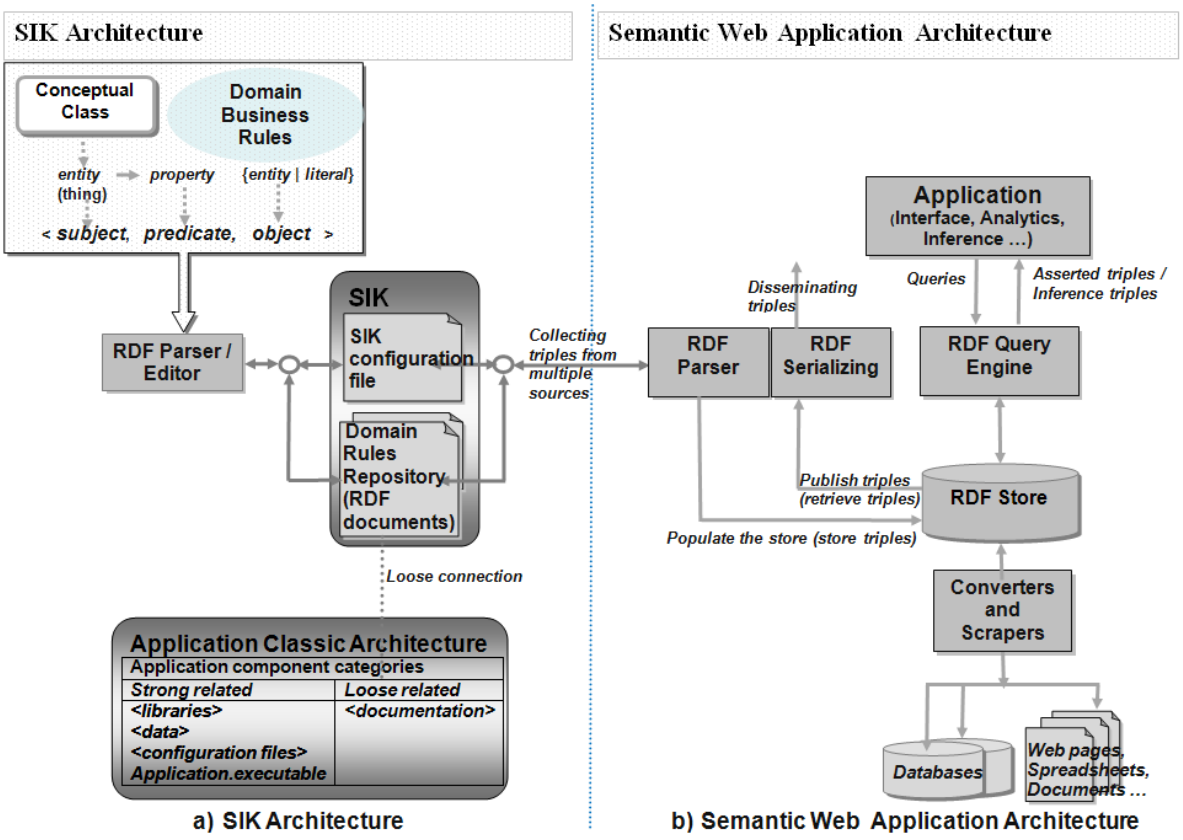

Fig. 5. SIK architectural component interaction with semantic web application [3]

The formalization of the business rules is realized through RDF/XML (Resource Description Framework/Extended Mark-up Language) serialized triples. A triple is defined by <*subject*, *predicate*, *object*>(Figure 5) as specified by RDF standard which was defined with the scope to represent and manipulate knowledge in web space. RDF usage is based on the specifications of the standard defined by the RDF Working Group of World Wide Web Consortium (W3C), the non-governmental organization, that defines and owners the Internet standards. The standard includes also the guidelines for Web architecture as defined by the creator of Internet Tim

Berners-Lee [8]. RDF is based on XML and in that way inherits all characteristics of that and adds his own characteristics. The SIK does not restrict the stored knowledge at the domain business rules. Being an RDF document, which in turn is based on XML, the SIK component has all the characteristics of both RDF and XML. SIK allows defining and keeping in the same document (file) a variety of rules grouped in separate spaces by intermediate of namespaces. It allows also to mix together protected rules (encrypted rules) and unprotected rules (unencrypted rules). The spaces also allow storing business rules regarding the application programming interface (used to connect applications over cyberspace) or the rules applied to operate the application in current use. SIK is a concept that identifies only a specialized RDF document: a document that collects mainly the domain knowledge incorporated in software products and user interaction knowledge, or in other words the business rules of a domain incorporated in software products. But SIK do not restrict to only these categories. It can collect any domain knowledge included or not in a software product. For business rules definitions and computation can be used the RuleML as described in [9] or we can use the Polish or Reverse Polish notation. In our case all business rules are very simple:

- The major part represented by recipes containing the preparation procedure and which is ease described in native RDF;
- Some simple formulas to compute the correspondence between different measurement scales or to realize equivalences and that are ease described by using the Polish/ Reverse Polish notation and that are also ease described in native RDF.

The services offered to consult the knowledge will allow the user to browse and see the content to which it is granted the access in a user friendly format. We can argue that both *Data Store* and *Knowledge Repository* act as a local database, respectively a local repository. They are parts of a large distributed database and distributed knowledge repository, respectively. They can be accessed, if they are online and their owner agree that, to determine if they need updates, if they contains news, if necessary to collect status information about user interaction, personalization, and experience in a similar way the distributed databases are queried and updated.

Data Analysis. A GUI based tool to access the *Data Store* and to realize a set of basic operations on the local data such as creating, updating, deleting or reporting. The tool allows also access to collected status information to ensure a full transparency to the process, access to the current value for setting parameters and options, etc.

Knowledge Extraction/ Inference. A GUI based tool to access the knowledge in the *Knowledge Repositories* for a "semantic search, and maybe, to infer new knowledge". This is very useful when the user has a lot of products available, in predefined quantities and want find out which recipes can be applied. This tool includes also processes to transparently track the user relevant activities and store them in the *Data Store* or in the *Knowledge Repository*. The component must be based on semantic web architecture, similar to the one described in Figure 5, right panel.

3.2.2 Server side components

The components at server side are briefly described in the following paragraphs.

Data Store. The server side collects all products and nutrition facts added by users. They require a human validation before be included permanently into the server *Data Store* and made available for all users. The validation refers to check the correctness of nutrition facts only for the base products since for recipes this is automatically determined from the constituents, generally stored here. At server side, a set of tables, are used to collect centrally statistics about user preferences. The behavior to the server side *Data Store* is similar with the one of a global schema for a distributed database with except that stores data as a centralized database. The global schema behavior allows query all

active local databases to determine if they require update or contains news to be added to the server side. All these operations take place only for the "white pages" like information and data, it means for the information and data that are not subject to copyright law and that cannot be delivered for free. *Data Store* is interfaced with system administrator to allow maintenance operations, validation and acceptance of facts, data analysis of collected status information, access restrictions to sensitive data etc. The final intention for the solution is to realize the database as XML documents and ensuring in that way the highest portability and availability for integration. In that way both data and knowledge will be stored, managed, and exploited using XML or constructs derived from XML, as RDF is.

Knowledge Repository. It contains all recipes definitions, knowledge about diabetes, nutrition facts, conversion procedures and formulas, equivalences between different measurement systems etc., or in other words the business rules of the domain subject or not of the copyright law. As for client side, the knowledge repository is interfaced with system administrator for creating new recipes definitions, for validating collected recipes, for protecting or freeing knowledge, or in other words all maintenance operation required by the knowledge. The creation of new recipes takes place by consulting the *Data Store* and then building the associated RDF definition to be stored as new knowledge. Once created a new recipe this one do not requires, when consulted, to access in turn the constituents in the *Data Store*. The server side *Data Store* will be consulted only in maintenance operations and/or to check for consistency of definitions. The services offered to consult the knowledge will allow the user to browse and see the content in a user friendly format. *Knowledge Repository* acts as a global or central knowledge repository and contains both unprotected (free of charge and deliverable to anyone wanting that) and protected knowledge (delivered only on a subscription basis). It contains also all domain knowledge incorporated in the Web services that contributes to the diet application functionality as a central cumulative SIK as defined in Avram [3] [4], and used for extensive knowledge management.

Web Services Repositories. This component contains the application web services that will be delivered to the individual users for free access together with all required tools to build and deploy a portable application. For that category of users, that invokes/ requests the services via the application provided by the server, the server acts as a *Service provider* and *Service broker*. After the individuals users defines for the first time the preferences for the application and any time they want change the application, these services are invoked from the server side and are "bind" in a portable application that in turn will be delivered to the user for the wanted platform. The Web services for business users will follow all the standards and rules for discovery and usage: they are published to a Service repository, are searched and found by a Service broker as an answer for a user request, and finally are bind to that user if they meet his requirements. In this way, for business user category, the server acts as a *Service provider* and competes, maybe, with other services provided by others for a contract.

User Management Services. This component contains all required tables and services for user management: creation and maintenance of users and user rights. It contains also the interface for user creation/ login allowing, depending on the category:

- Individuals users: access to the steps 1 and 2 as defined in "The individual users interaction" chapter in that paper;
- Business users: access to the same services as individuals together with specific services such as online payments, invoices management etc.

It contains also the services for authentication and for user access rights checks ensuring the first level of protection for data and knowledge.

Security services. This comprises all services required to protect the information subject to subscription against unauthorized access. They allow distinguish between protected and unprotected information in all the steps of user interaction with the server itself, Data Store, Web Services Repository, Knowledge Repository, and User Management Services. The Security Services acts as a firewall protecting all sensitive data or knowledge from user actions and programs/ agents actions. It allows also the proper delivery of services.

3.3 The individual users' interaction

The main steps of user interaction with the proposed solution can be categorized as *Definition* (1) and *Utilization* (2), as illustrated in Figure 6.

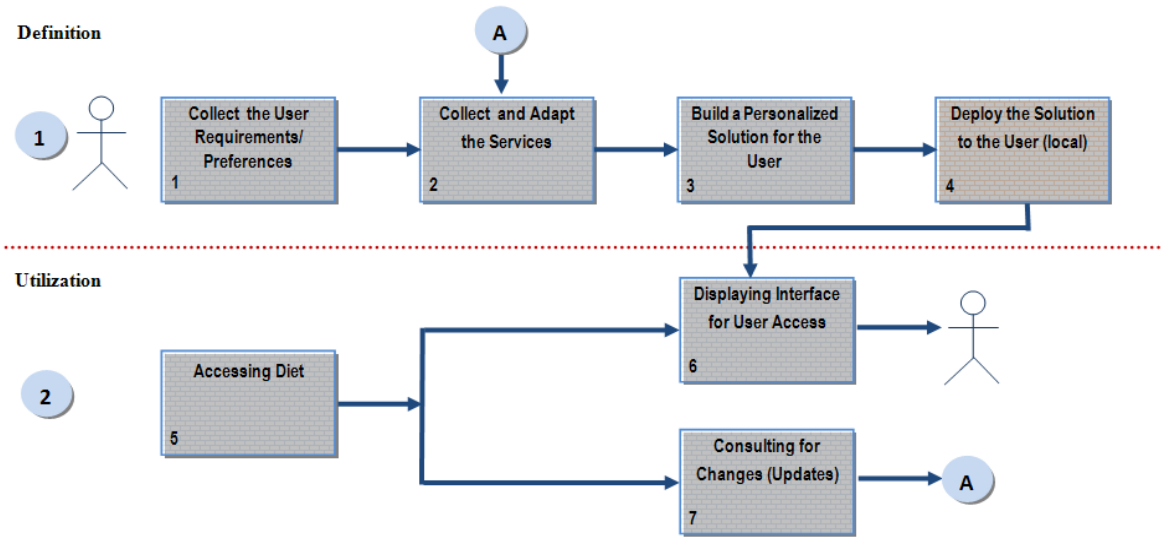

Fig. 6. The main steps of user interaction

The *Definition* main steps are:

- *Collect the User Requirements/ Preferences*: in which the user requirements/ preferences are collected by following a menu-driven interface. All these user requirements/ preferences will be stored locally in the user computer (workstation) and, after the completion and acceptance by user will be used to collect statistical information that will be used to optimize the product functionality and to establish a default state for the user category/ region. The information can be used later by user to specify his changes and new requirements in the same menu-driven interface where the default values will be the ones previously given to the system;
- *Collect and Adapt the Services*: in which on the basis of user requirement/ preferences will be generated the requests for user interface services that will be passed to broker (s) to find them. The difference here from the business Web services is that the solution here is for a free of charge support and in that situation do not expect be to many offers for services. Thus the broker will be provided, in the first deployment, by the service provider server with the scope to select and offer the best (version of the) service that meets the processing needs of the user;
- *Build a Personalized Solution for the User*: in this step is build a personalized solution for the user interface, from adapted and chosen services, in a similar way traditional standalone applications are, that will be the initiator for all other services invoked at runtime by the user interactions with the application. Here is not excluded the generation of the entire application as a standalone/ portable application version. What is new here is that the services are incorporated in the user interface so their functionality remains unchanged until the

user decides to update them. This way described here differs from the general functionality of binding in which the binding information is used to invoke the Web service from the web service provider that publishes the chosen web service. Generally Web services implemented as objects and the invocation is realized by calling a method of the chosen object. In turn the method can invoke other methods, available or not in the public interface, of the same object or of other objects if the Web service is build from other Web services (using orchestration and choreography, for example). By building a "standalone" like solution all involved objects must be included in that in a similar way the external modules included in a standalone application program at link-edit stage;

- *Deploy the Solution to the User (local)*: the personalized solution is deployed to the user as a standalone/ portable user interface application and as the data and knowledge recipients (data store and knowledge repositories). The portable application, in Microsoft Windows environment, for example, contains all needed pieces included, such as library modules and do not requires to install that libraries and declare them in the registry. The portable solution is realized by tools at server side. Combined with a deployment of the Data Storage component as XML files and knowledge repositories as a SIK component, the process takes place more like a file transfer from server computer to client computer than an installation procedure in the classic sense. In the same way depending on the case, first time used or current exploitation of the application, the solution will be deployed as an initiator (first time installation) or as an upgrade (current usage). In that way we have the possibility to solve in an ease to use way the deployment even for "dummy" users may have. The user solution must provide the mechanisms to check and determine if necessary to incorporate the new or updated services available for a specific purpose. This must be realized in accordance with the specification of services requests, defined by the initial specification and any other updates of that. All these processes must be realized in an automatic manner and the user will be implied only to accept or reject the update, described and motivated in non- or technical terms depending on the way the user specifies in the setup parameters.

The *Definition* main steps are triggered by creating the user account and requesting the application. After that, the remake of the application can be expressly requested, by accessing the user account or indirectly, by accessing the application for current operation by *Accessing Diet* (step 5). The step five is the one in which the users access the application in the local computer for current operation. The update finder service will determine if updates where available and if yes and user wants that it goes to *Consulting Changes (Update)* and then initiate the step *Collect and Adapt Service* and followings. If no update exists or accepted the system goes and *Display Interface for User Access.* The user can operate and use now the application for his processing, such as:

- It can consult the knowledge stored in repositories in a similar way it consults a product catalog;
- It can consult a cookbook like containing the recipes;
- It can learn about the application operation;
- It can define new recipes, maintain Products catalog, or can print the available reports;
- It can compute nutrition facts for a specific recipe;
- It can determine the nutrition facts for new defined recipes or for the existing ones in which it changes the quantities or replaces the constituents, etc.

In fact this is the most used step for almost users: they cook the food and uses different quantities of the constituents. The product will cumulate from these quantities of products and associated nutrition facts from the product catalog and determine the values for that. Later on these cumulative values will be divided to the final quantity of

prepared food and will be able to find the composition in nutrients per serving or can determine which is the quantity per serving with respect for restrictions of some nutrients quantity, for example carbohydrates less than 50 gr. The solution, offered here, do not determine if the components, when applying a recipe, will be transformed from one nutrient in another one by different chemical reactions that can be resulted as the cooking procedure applied. This information must be supplied by the cooking procedure in the recipe and must be applied later as a correction to the cumulative value of nutrients to determine their approximate real value.

4 Conclusions

The main change in the way the Web services used refers here to the possibility to build a freeze user interface from Web services and to make from that an initiator for all other web services the application may invoke. It is not excluded the solution to build an entire standalone/ portable application able to offer to users the possibility to access the basic functions of the application in an offline way. The portable/ standalone solution is more efficient and answer better to the requirements since his deployment is more like a copy to a specific storage than installation.

Another change is to realize a reduction of required storage space and processing at server side by deploying both data and domain knowledge to user side with ensuring the agility. The domain knowledge will be deployed as a SIK repository so that software will be able to infer new knowledge and realize semantic searches as described in reference [3]. Both data store and knowledge repository will have local and global behaviors similar to distributed databases. This will allow to disseminate data and knowledge and later on to enrich the content with user creations like in Hegel's dialectic. Because the knowledge stored in SIK is process-able, we can use inference engines to infer new knowledge from it, that is, to find new rules. It can be used also as a basic source for semantic search. SIK is a concept that identifies a specialized RDF document: a document that collects mainly the domain knowledge incorporated in software and user interaction knowledge, or in other words the business rules for a domain. It was defined to signal a way to use the "tacit" like knowledge incorporated in the software products and to allow users of that software to inherit that incorporated knowledge. But SIK do not restrict to only these categories of knowledge. It can collect any domain knowledge included or not in a software product. For business rules definitions and computation can be used the RuleML [9] or we can use the Polish or Reverse Polish notation. One of the main benefits of representing in this formal way the business rules is represented by that these rules become automatable: they can be processed and acted by business rules engines, can be searched by semantic search engines, and inference engines can infer new knowledge from the existing ones.

The application manages both, protected and unprotected, data and knowledge. The application must deserve two broad categories of users: one having free access to public data and services, it means to unprotected information and knowledge, and another one with paid services, having access to both protected and unprotected data and knowledge.

The solution, offered here, do not determine if the components, when applying a recipe, will be transformed from one nutrient in another one. This information must be supplied by the cooking procedure in the recipe and must be applied later as a correction to the cumulative value of nutrients to determine their value. Despite that the solution gives a closed approximation to the nutrients and signals the user their values. These are real flags signaling what is safe for his health with respect for a specific diet.

One major advantage is that the solution offers information about cooking and food without boundaries, habits or cultural restrictions etc. It transforms that specific

knowledge to a universal knowledge that can be shared without boundaries, religion, or social regime restrictions and constraints: it can offer, for a specific diet, a plethora of recipes from around the world. In the same time it offers to the user the possibility to adapt and transform them according to their behavior and taste with keeping the possibility to control the quantities in the monitored nutrients of his final product. Of course the only solution to know exactly a food contains is to realize chemical analyzes. But including that chemical analyzes will produce approximate values if the cooking is realized manually and not in an industrial way controlled by machines, and where we suppose that all steps will be followed exactly and the temperature will not overcome. I consider that is satisfactory for the target audience that prepare their food in a manually manner.

The database schema for the *Data Store* component in the architecture includes a 'non academic" solution in that the *Products* table contains columns corresponding to the basic nutrition facts (even they are 0 for some products) and allows extend these facts in an additional table. These additional facts can be exploited by stored procedures or access routines provided by database administrator with the scope be easy accessed by services. This solution can be eliminated with an "academic" one represented by an XML database and ensuring in that way the highest level of portability and integration that can be ensured by the today technologies.

For all business users and other *Service requestors* of a Web service from the ones included here the web services will be available according to the traditional way described in [3].

As a final conclusion the solution realizes a description of the main architectural components. In different parts some details included with the scope to certify to learner that the solution is feasible and can be realized an implemented in its integrality.

Even parts of the components described earlier are realized this document is not the description of an informatics system or product. It can be considered as a strategic specification for both, an informatics system and of another way to use services and build applications, using those services, as portable applications.

References

[1] S. Albin, *The Art of Software Architecture: Design Methods and Techniques*, John Wiley & Sons, Books24x7 <http://common.books24x7.com/book/id_6020/book.asp>, 2003, pp 1-20.

[2] ANSI/IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 2000, pp 1-29.

[3] V. Avram, "The Acquisition and Sharing of Domain Knowledge Contained in Software with a Compliant SIK Architecture", *Proceedings of the 11th European Conference on Knowledge Management*, Universida de Lusíada de Vila Nova de Famalicão, Famalicão, Portugal, VOLS 1 AND 2, 2010, pp 19-26.

[4] V. Avram, "Benefits of Using Software with SIK Compliant Architecture in Cloud", *5th International Conference Knowledge Management - Projects, Systems and Technologies*, Academy of Economic Studies and the National Defense University 'Carol I', ISBN: 978-973-663-783-4, Bucharest, Romania, 2010, pp 75-78.

[5] V. Avram, D. Avram, "Transforming the Knowledge Incorporated in e-Learning Software into an Automatable Explicit Knowledge for the Teacher and the Learner", *Procedia - Social and Behavioral Sciences*, Volume 11, Elsevier, 2011, pp 180-184.

[6] M.A. Babar and I. Gorton, "Architecture Knowledge Management: Challenges, Approaches, and Tools", *29th International Conference on Software*

Engineering (ICSE'07 Companion), 0-7695-2892-9/07 IEEE Computer Society, 2007, pp 170-171.

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Second Edition, Addison-Wesley Professional, http://acmsel. safaribooksonline.com/ 0321154959, 2003, pp 1-17.

[8] T. Berners-Lee, D. Connollyand R. Swick. (1999, June 7).Web Architecture: Describing and Exchanging Data, W3C Note 7 June 1999. *World Wide Web Consortium (W3C)*. [Online] Retrieved October 10, 2010, from http://www.w3.org/1999/04/WebData, 1999.

[9] H. Boley, J. Mei, M. Sintekand G. Wagner. (2005, April, 27-28).RDF/RuleML Interoperability, *W3C Workshop on Rule Languages for Interoperability* [Online] Position Paper: 27-28 April 2005, http://www.w3.org/2004/12/rules-ws/paper/93/.

[10] B. Bos. (1997, July, 11). XML representation of a relational database, *W3C - World Wide Web Consortium*, [Online] http://www.w3.org/XML/RDB.html.

[11] I. Gorton, *Essential Software Architecture*, Springer -Verlag Berlin Heidelberg, 2006.

[12] G. Coulouris, J. Dollimore and T. Kindberg, *Distributed Systems Concepts and Design*, Third Edition, Addison-Wesley, Pearson Educational, 2001, pp 1-720.

[13] J. Hurwitz, R. Blooc and M. Kaufman, *Service Oriented Architecture for Dummies*, John Willey and Sons, 2007, pp 1-408.

[14] P. Seshan,*Proces-Centric Architecture for Enterprise Software Systems*, CRC Press, Boca Raton London New York, 2010,pp 1-304.

[15] P. Watkins,*ABC of Diabetes*, Fifth Edition, BMJ Books, 2003, pp 1-18.

[16] WHO – World Health Organization. (2011, January). Media Centre->Factsheets, *Fact sheet N°312January 2011*, [Online] http://www.who.int/ mediacentre/factsheets/fs312/en/.

Vasile AVRAM PhD has graduated the faculty of Economic Computation and Economic Cybernetics in 1976. Now is professor of Internet Technologies for Business and Informatics for Business Administration at Faculty of Business Administration from the Academy of Economic Studies in Bucharest, Romania. His research interests mainly include Internet Technologies, Database Management Systems, and Knowledge Management. Is member of the professional associations IEEE, ACM, INFOREC, and IACSIT.

Diana AVRAM has graduated the Faculty of Mathematics to the University of Bucharest in 1995. Actually is web-designer and webmasterat SIGMA PublishingCo., Romania. The research interests mainly include Internet Technologies, Database Management Systems, and Knowledge Management.