The Personalization Server as a Vital Component of any CRM Application. Considerations on a Promising Software Product

Adapted Services

eGO is a personalization engine that uses the history of user interaction with the eGO client for building dynamic user profiles in real time.

The system can handle two types of users:

- **registered users** for which a history is maintained in order to maximize the accuracy of the estimated preferences
- **temporary users** for which only the current session is used for personalization

eGO might be used for e-Commerce solutions or by any web solution for personalizing the information sent to the users.

Software Architecture

The server was implemented in Java and therefore eGO might run on every operating system that provides a JDK 1.4 implementation. The eGO might be run distributively on several machines in order to increase the overall performance. The communication between distributed components is done by using RMI.

Communication between the eGO and the eGO client is done by using a SQL database. Currently eGO can work with PostgreSQL and MySQL. Support for other SQL servers is possible in order to suit individual requirements. The design also allows for a easy adaptation of eGO in order to use other databases such as LDAP.

eGO was implement by taking into account the following design guidelines:

- minimizing the usage of resources (CPU, memory);
- distributing and multithreading at the computational level;
- localization;
- scalability as to the number of users and the hardware resources required;
- quick convergence of the algorithms used to compute the user profiles;
- support for both RMI and CORBA;
- encapsulating the database code in order to switch easily to other databases.

The system is built around the following basic components:

- **The Interests Component** primarily processes the interests generated by the users
- **The History Component** is responsible for storing the processed interests by maintaining a history for each user
- **The Assistant Component** provides direct estimations of users' preferences by processing the user histories
- **The Log Component** is a log server shared by all other components
- **The Ontology Component** manages the classification over the whole range of proposed topics
- **The Cleaner Component** is responsible for removing the old and temporary users that have become useless for the system

A series of optional components are used to increase the accuracy of user profile determination:

- **The Collaborative Estimator** uses a collaborative filtering technique for estimating the user's interests by comparing those interests to the interests of similar users
- **The Inference Estimator** can add preferences to the user profiles working on the assumption that with similar topics several users would have similar interests
- **The Rule Estimator** adds preferences to the user profiles by using rules that define the correlation between various topics. Since no history is available for temporary users, their profiles are provided by the optional estimators.

Besides these components, the eGO Administration Server allows for easy administration by using a Web browser.

In order to make eGO running, the information managed by the eGO client should be classified. The result of the classification should be a set of representative topics (structured as a tree). Each indivisible piece of information from the eGO client (e.g. a web page) should have attached a set of topics that are representative for it.

In order to build user profiles, eGO needs the history of user interaction with the eGO client (Adaptive Server). This is done by the Adaptive server through sending interests to eGO each time when a client requests information (documents). The interests are simply records in a SQL table.

The results of eGO computations are the user profiles. Each profile contains in a set of assumption regarding the user preferences that physically consists in a set of records within the profile_preferences table. Each record specifies the user interest for a topic.

System Requirements

- PostgreSQL 7.1 or MySQL 3.2 (others SQL servers on request)
- Java2 v 1.4
- JSP-compliant web server (e.g. Tomcat 3.2) for running the administration application
- Linux (others on request)
- 1 GHz Intel Pentium TM Processor
- 512 MB RAM
- 10 GB Hard Drive

How it works

Introduction

eGO is a personalization engine that uses the history of user interaction with the eGO client for building dynamic user profiles in real time.

The server was implemented in Java and therefore eGO might run one every operating system that provides a JDK 1.4 implementation. The eGO might be run distributively on several machines in order to increase the overall performance. The communication between the distributed components is achieved by using RMI.

Communication between the eGO and the eGO client is done by using a SQL database. Currently eGO works with PostgreSQL and MySQL. It is also possible to suit individual requirements support for other SQL servers. The design also allows for a easy adaptation of eGO in order to use other databases such as LDAP.

1.Internal Architecture

eGO was implemented by taking into account the following design guidelines:

- minimizing the usage of resources (CPU, memory);
- distributing and multithreading at the computational level;
- localization;
- scalability as to the number of users and the hardware resources used;
- quick convergence of the algorithms used to compute the user profiles;
- support for both RMI and CORBA;
- encapsulating the database code in order to easily switch to other databases.

The system is built around the following basic components:

- **The Interests Component** primarily processes the interests generated by the users
- **The History Component** is responsible for storing the processed interests by maintaining a history for each user
- **The Assistant Component** provides direct estimations of users' preferences by processing user histories
- **The Log Component** is a log server shared by all other components
- **The Ontology Component** manages the classification over the whole range of proposed topics
- **The Cleaner Component** is responsible for removing the old and temporary users that have become useless for the system

A series of optional components is used to increase the accuracy of user profile determination:

- **The Collaborative Estimator** uses a collaborative filtering technique for estimating the user's interests by comparing those interests to the interests of similar user
- **The Inference Estimator** can add preferences to the user profiles working on the assumption that with similar topics several users would have similar interests
- **The Rule Estimator** adds preferences to the user profiles by using rules that define the correlation between various topics. Since no history is available for temporary users, their profiles are provided by the optional estimators.

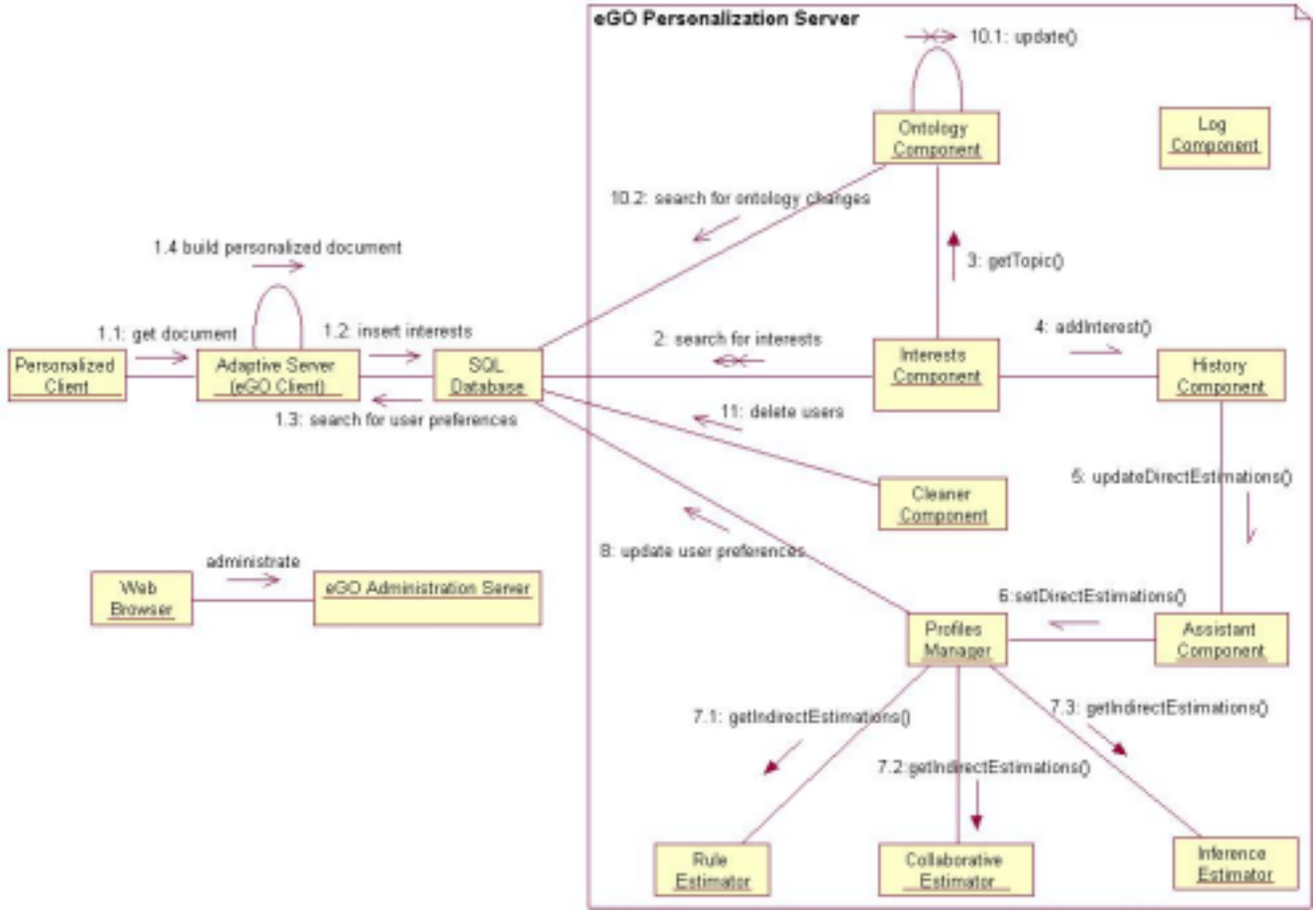
A supplementary component, a web-browser-based server takes care of easy administration.

Each component may run on a separate Java virtual machine (JVM). This enables eGO to use all resources available in order to increase overall performance. The JVMs may run on separate computers or they may run even on the same machine (to deal with the 2/4 GB memory limitation for a JVM).

The design took into account such future improvements as:

- using the overnight time (or any time available) of other computers;
- distributing evenly the individual components.

The next UML diagram depicts the collaboration between the eGO components and between eGO and the eGO client.



Ontology

In order to enable eGO to run, the information managed by the eGO client should be classified. The result of the classification should be a set of representative topics (structured as a tree). Each indivisible piece of information from the eGO client (e.g. a web page) should have attached a set of topics that are representative for it.

The **Ontology Component** caches these topics tree and builds indexes in order to provide fast ontology access to each eGO component. one therefore The ontology components have the supplementary task of periodically updating the ontology, thus implementing the dynamic ontology concept.

How does eGO receive information?

In order to build user profiles, eGO needs the history of user interaction with the eGO client (Adaptive Server). This is done by the adaptive server through sending interests to eGO each time when a client requests information (documents). The interests are simply records in a SQL table and they contain:

- **the userID** which uniquely identifies the user;
- **a keyword to** specify the topic related to the requested document;
- **a rank** (a double between 0 and 1) specifies the user interest for the submitted keyword;
- **confidence** (a double between 0 and 1) allows the eGO client to specify his confidence in the interest submitted;

- **source** is a text that should identify the source of the interests (for instance it might be the PHP script file name).

How does eGO process information?

The **Interests Component** is responsible for periodically searching for new interest records. It primarily processes these interests (validations, finding the topic that matches the keyword) and then sends the interests to the history component.

The **History Component** is responsible for storing the interests received from all users. In order to reduce the amount of information stored for each user, the History Component compresses the interests by using an algorithm that provides enough information for the Assistant Component in order to quickly compute accurate estimations of the user preferences.

The **Assistant Component** processes the user history and computes direct estimations for the topics for which the user generated interests. The estimations are sent to the Profile Manager Component.

The **Profile Manager Component** is responsible for updating the user profiles within the profile_preferences table. Because the direct estimation might not be enough (the user is new and he generates interests only for very few topics), the Profile Manager might use the optional estimators for completing the user profile with indirect estimations as we already mentioned.

What are the results?

The results of the eGO computations are the user profiles. Each profile contains a set of assumptions regarding the user preferences that physically consist in a set of records within the profile_preferences table. Each record specifies the user interest in a topic:

- **userID** uniquely identifies the user;
- **topicID** identifies the topic;
- **rank** (a double between 0 and 1) is the estimation for the user preference for the related topic;
- **confidence** (a double between 0 and 1) is the eGO confidence in the submitted interest (this is bigger for direct estimations and smaller for indirect estimations);

The eGO client might use these preferences for adapting the information sent afterwards to its client.

eGO Use Cases

2. Creating a user profile

In order to add a new user (create a new user profile) into the ego server, you will have to add a new record to the profile_users table. This table contains only the basic attributes needed by ego in order to operate on the user profile:

- userID (should be unique)
- userTypeID - a flag marking the user type: deleted (0), persistent/registered user/client (1), temporary user (2)
- updateTime - the time stamp of the last update of the user profile.

If one decides to use eGO tables to store additional information about a user, this is the place where the attributes should be added (like 'login', 'password', 'pin', etc.). The modification of the table diagrams can be done before creating the database by modifying the script that creates the tables, or after eGO is installed by calling 'ALTER TABLE' SQL commands on 'profile_users' table.

One may also keep this kind of information in a separate table, or even in a separate database.

Examples (PostgreSQL SQL script):

Ex 1. Adding a registered user with additional login name and password:

```
INSERT INTO profile_users (userID, userTypeID, login, password) VALUES (4343, 1, 'first_customer', 'secret');
```

Ex 2. adding a temporary / occasional user with an additional sessionID:

```
INSERT INTO profile_users (userID, userTypeID, sessionID) VALUES (34234, 2, '9098vsdf0dff0v');
```

3. Login

The responsability of logging in the user falls on the ego client side. Logging in a user means usually identifying the id of the user by knowing some other information about her (some of the information may be secret). E.g. login / password.

These supplementary information may be stored in the 'profile_users' table belonging to the ego database, or anywhere else, as the ego client decides. The ego client is responsable for storing the right information that will enable him to get the user id ('userID' field in the 'profile_users' table) by knowing, for example, the login name and the password.

Examples (PostgreSQL SQL script):

Ex 1. Looking for the user id with the login name 'first_client' and password 'secret'; this will return the userID if the login and password are correct, empty set otherwise:

```
SELECT userID FROM profile_users WHERE login = 'first_client' AND password = 'secret';
```

4. Adding an interest to the user profile

There are four elements that define an interest on a specific topic regarding a given user:

- - userID - the ID of the user for which the interest is appended; this ID should be one of the user IDs found in the 'profile_users' table;
- - keyword - a keyword denoting the topic; it should be one of the keywords stored in the 'ontology_keywords' table;
- - rank - the amount of interest in the topic: 0 means no interests, 1 total interest;
- - confidence - the confidence in the interests of the user in the specified topic: 0 means no confidence at all, 1 total confidence.
- - source (optional) - denotes the entity that generated the interest.

Adding such interests to the user profile is done by adding a new record to the 'profile_preferences' table.

Examples (PostgreSQL SQL script):

Ex 1. Adding a big interest for the user with ID 4343 in topic 'sport' with total confidence (as specified by the user in a registration form):

```
INSERT INTO profile_preferences (userID, keyword, rank, confidece, source) VALUES (4343, 'sport', 0.95, 1, 'registration');
```

Ex 2. Adding interests for the same user and topic, when the user chooses to print a document on the topic:

```
INSERT INTO profile_preferences (userID, keyword, rank, confidece, source) VALUES (4343, 'sport', 0.80, 0.7, 'print');
```

Ex 3. Adding interests for the same user and topic, when the user browsed a page on the topic:

```
INSERT INTO profile_preferences (userID, keyword, rank, confidece) VALUES (4343, 'sport, 0.70, 0.23);
```

5. Querying for the user profile

One can retrieve the user profile by just looking into the 'profile_preferences' table.

Examples (PostgreSQL SQL script):

Ex.1 Retrieving all the interests for a user with ID 4343:

```
SELECT topicID, rank, confidence FROM profile_preferences WHERE userID=4343;
```

Ex 2. Retrieving only the interests that are above average in rank:

```
SELECT topicID, rank, confidence FROM profile_preferences WHERE userID=4343 AND rank > 0.5;
```

Ex 3. Retrieving the first 5 best interests for the user with ID 4343:

```
SELECT topicID, rank, confidence, rank*confidence AS prod FROM profile_preferences WHERE userID=4343 ORDERED BY
```

prod DESC LIMIT 5;

6. Extracting statistics from the user profiles

One can extract all sorts of statistical information by just querying the 'profile_preferences' table.

Examples (PostgreSQL SQL script):

Ex 1. retrieving the top 50 best overall interests:

```
SELECT topicID, rank, confidence, rank*confidence AS prod FROM profile_preferences ORDERED BY prod DESC LIMIT 50;
```

7. Deleting a user profile

The deletion of the user profile is performed by a special component of Ego. In order to delete a user profile, the ego client must mark the user as 'deleted', by changing her type from 'registered' to 'deleted'.

Examples (PostgreSQL SQL script):

Ex 1. Deleting the profile for the user with ID 4343:

```
UPDATE profile_users SET userTypeID=0 WHERE userID=4343;
```

8. Expanding the ontology

The ontology may be expanded in two ways:

- adding a new topic to the 'ontology_topics' table. A topic has:
  - a unique ID (topicID),
  - a name (name),
  - the ID of the parent topic (parentID); if there is no parent, this value is 0.
- adding a new keyword to an existing topic in the 'ontology_keywords' table. A record in this table has two elements:
  - a name denoting the keyword (name);
  - the ID of the topic that this keyword belongs to (topicID); this value must be one from the 'ontology_keywords'.

Examples (PostgreSQL SQL script):

Ex 1. Adding a new topic, 'football' with topic ID 9, under the topic 'sport' having topic ID 4:

```
INSERT INTO ontology_topics (topicID, name, parentID) VALUES (9, 'football', 4);
```

Ex 2. Adding a new keyword ('soccer') for the topic 'football':

```
INSERT INTO ontology_topics (topicID, name) VALUES (9, 'soccer');
```

Traian Ocneanu
Director, NOVUM Computing SRL of Iasi, Roumania
to@novum.ro
www.novum.ro

Submitted on 20th November 2002 to SSGRR winter conference in ITALY