Meta-Modeling *Relevance Feedback* process with ATOM^3 Diseño de Software basado en modelado y simulación

Alejandro Bellogín Kouki
alejandro.bellogin@uam.es

February 4, 2008

1 Introduction

In the field of Information Retrieval one of the biggest problems is to know which is the goal of the user, her information needs, but sometimes she does not even know which are these needs. This is because a lot of reasons, for example the user has little knowledge of the collection or the retrieval system, perhaps she does not know how to utilize the query language used by the system (natural language versus bag of words), furthermore query ambiguity does exist because of synonymy and hyponymy (i.e., *python* may refers to the snake or to the programming language).

From all this one conclusion arises: creating good search queries is hard. A possible solution to this problem is the query reformulation, a manual process which requires quite a lot of intellectual effort. The relevance feedback process comes from making the query reformulation a controlled and automatic process.

In the next section we will explain in detail this process, how we have modeled it using the ATOM^3 tool and some advantages we have obtained from this (mainly simulation and code generation).

2 Proposal

This work started with the meta-model of the relevance feedback process, this meta-model is enough general to create a useful domain-specific visual language and quite simple to be understandable by an ATOM^3-experienced user who wants to know how this process works (and play with its parameters) or wants to show to other people its workflow.

In figure 1 we can see the class diagram used in this proposal. In section 2.3 there is a more concrete definition of relevance feedback, with its possibilities and history. Now we are going to focus on the (meta-)modeling process and in sections 2.1 and 2.2 we will show two features based on this model and made possible thanks to the use of ATOM^3.

Although the presented model is very general and it would allow a lot of different configurations, we are interested in the explicit case, which is the one in which the user can actively be involved and, because of that, is very appropriate for the simulation task. Besides that, there are some issues we can not achieve for the sake of simplicity and completeness (see section 3).

Primarily, this model is used to generate some code that the mentioned tool ATOM^3 presents to the user in a more friendly way, as buttons and/or images (see figure 2). We have to notice that the abstract

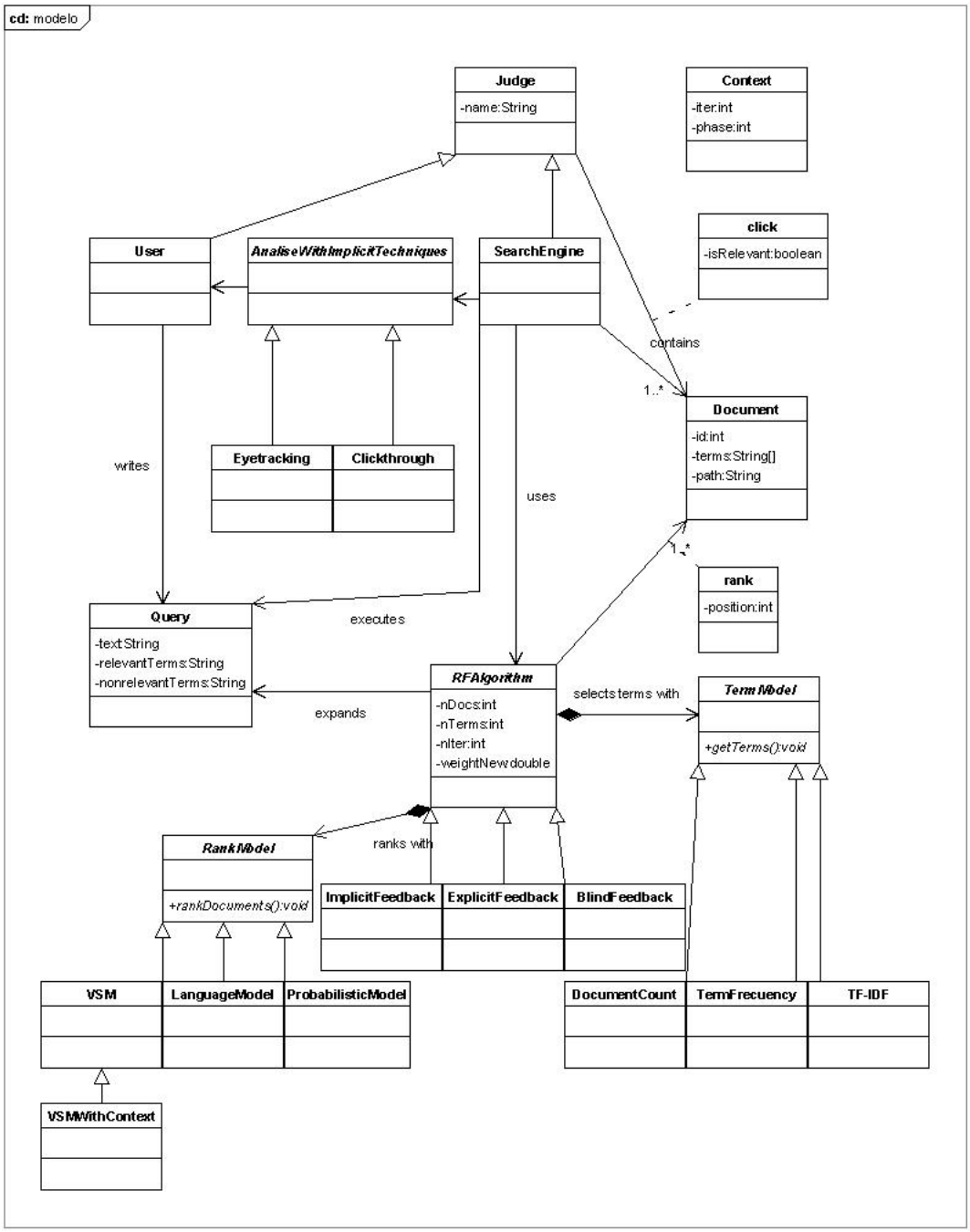

Figure 1: Class diagram for the relevance feedback model

classes are not allowed to be generated, because they are not instantiable. With these buttons the user can create her own model[1], with the condition that this model has to be conformed to the meta-model

[1]This is done loading as a meta-model the file `RF.py` generated automatically by ATOM^3 from the file `RF_CD_mdl.py`, in

presented previously. This is done automatically by ATOM^3, checking all the constraints included in the model (cardinality, inheritance, connectivity). Here we have one of the reasons why it is so important to have a complete and perfect (but simplified) model about the process to study, because it could produce inconsistencies or undesirable situations in its instances.

Figure 2: ATOM^3 presents these buttons to the user to make her own models

Once the user has created her model (as an instance of the meta-model presented in figure 1) she can do a simulation or generate code if it is enough specific (i.e. there must be some documents in the model). Now we are going to explain how these tasks can be carried out, always based on the principles of relevance feedback (section 2.3) within the presented framework (ATOM^3 and meta-model as in figure 1).

2.1 Simulation

This task is very interactive and it is intented to be very interesting for the final user, because she can try different configurations and test, in a primitively way due to the simplications made, whether these configurations produce the desired effects or not, furthermore she can do all this without writing a line and with a very intuitive interface, specially designed for this problem. Moreover, if she wants to compare these results with the real ones (because she is satisfied with the results or they are a bit strange) she only has to generate the code and try it for herself.

Here we have a list of the different features adjustable in this framework[2]:

Type of algorithm You can choose between the different kinds of the relevant feedback algorithm: explicit, implicit, blind.

Ranking model This is the model used to rank the documents according to the query, the choices are: vector space model (VSM), language model, probabilistic model and VSM with context[2].

Term model This model says how the terms are selected within the documents, this process can be done attending to the term frecuency in a document, the number of times a term appears in all the documents or with a combination of both (tf·idf factor, term frecuency and inverse document frecuency).

Number of iterations This is the number of times the simulation will run as maximum.

which we have created the meta-model formerly

[2]By the moment it is only implemented the first one: vector space model

Number of terms This is the number of terms that will be chosen from each document.

Weight for the new terms The new terms added to the query will have a fixed weight, defined by this parameter.

Parameters of the feedback formula Other parameters required in the formula 1 (a, b, c).

Collection and query You can choose which is the query the system will execute, and against which set of documents.

For the simulation we have also had to define some graph grammars, if we assume the instance start with the phase set to SEARCH, since the first phase (INIT) is provided to define the documents and their relations, the following are the rules we have used in this simulation:

1. First rules are the ones which rank the documents according to the query, they have knowledge about the terms the documents contain and the formulated query, depending on the type of algorithm and the choice of the model to rank and choose the terms, the results (ranking) may vary. There are two different rules because when the feedback step has finished and the system start again we have to delete the previous relevance assessments given by the user. We can see them in figure 3 with their corresponding Negative Application Constraint (NAC) to avoid the infinite loop this rule would cause, because if this NAC would not exist the left hand side of both rules could always be executed.

2. The next rule by priority is the one which changes the state of the system (phase), it changes from SEARCH to FEEDBACK, with the meaning that now it is the turn to the user to vote which documents are relevants for her and which not.

3. The next two rules are equivalents and represent a user who votes for a document to be relevant or not. We can see them in figure 4. They also have a NAC to avoid an infinite loop as before.

4. The last rule to be executed by priority is the one which checks if the number of iterations has been reached, and in other case it changes the phase to SEARCH, modifies the query according to the algorithm and starts again.

Now we are going to explain the steps to follow to run the simulation properly.

2.1.1 Steps to simulate

1. First of all, we have to do a model or instance of our meta-model, for this example we are going to use the one already created and with the name `ex_RF_VSM_TFIDF_mdl.py` (in this example we have modeled a system with a user and a query defined, five documents and a search engine using an explicit feedback algorithm which uses as rank model the VSM and as term model tf·idf). We open it as a model. We have to see something like the figure 5.

2. Then, we go to *Transformation* menu and click on *Execute transformation*, after clicking in the new button we select the file `gram_EF_VSM_TFIDF.py` (it contains the grammar rules generated from the file `gram_EF_VSM_TFIDF_mdl.py`). We can add other grammars, but this example will only work for this specific grammar. It is recommended to tick the next options: `STEPbySTEP`, `Sequential Manual` (in `Execution`) and `Dangling Condition`, besides to let ticked the `Injective` option.

3. When we click in the `Ok` button, we will see a new window telling us which grammar is being executed, which is the last executed rule and three buttons more: `Step`, `Continuous` and `Close`. We will use the first and last buttons, the former will let us to move step by step in the simulation and choose which object is going to interact with the system, the latter will end the simulation. Some steps are shown in figures 6, 7, 8.

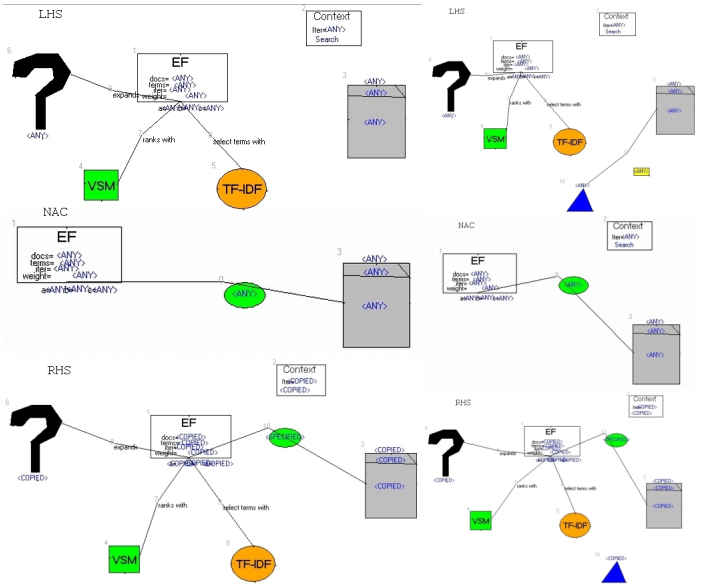

Figure 3: First two rules to be executed (if possible)

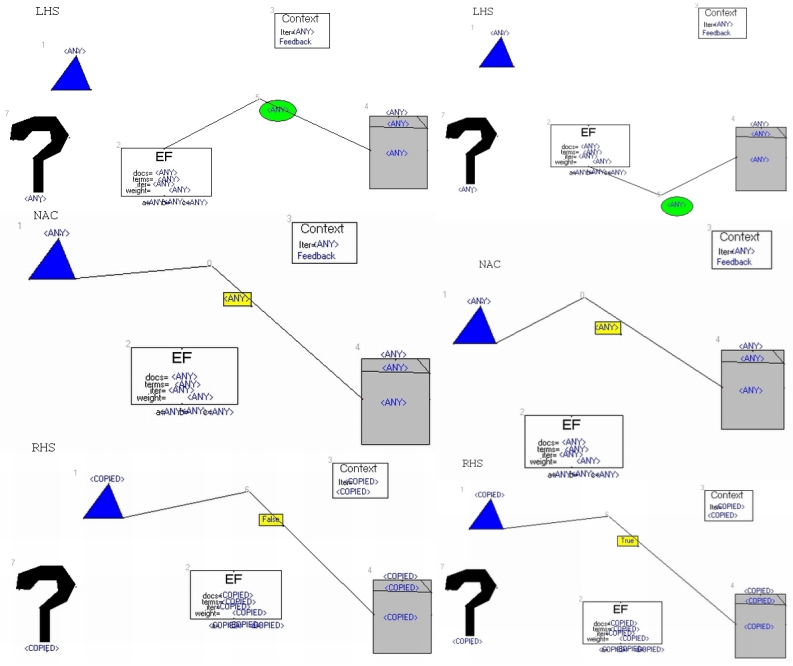

Figure 4: Two rules that represent a vote for the user

We have to say that it is possible to change this example however we want, you can modify any of the parameters listed before and run the simulation in that situation.

2.2 Code generation

Besides simulation, in this work we have tried to generate code from a model, instance of our original meta-model. For this task, we have used the Terrier[3] platform to get a program which indexes documents and retrieves them when a query is given, following the relevance feedback process.

Unfortunately we were not able to complete this program on time, but we do generate code in the form of a configuration file using some declared functions in `RFCodeGeneratorHelper.py`. In this way, the user has to create a model in ATOM[3] and, if it is valid, a configuration file is generated when she presses the `Generate code` button. If she instances a query object, the system will use this query as an input, and in other case it will ask for one.

We have to note that the generated configuration file is like the ones used by Terrier, but we did not succeed in developing a program acting like the relevance feedback process (we were able to index the documents and to retrieve results, but not to repeat the process). In any case, we think this is one of the better ways to achieve the problem of generating code: create something that the user can use without touch anything and plug it into other program, also unknown for the user.

2.3 Relevance feedback

For a better comprehension of this work it is necessary to understand the relevance feedback process, its importance and scope.

The relevance feedback process was introduced in the 1960s as a controlled, automatic process for query reformulation [5]. The query formulation tries to solve the problem that users normally use queries that consist of 1 to 3 terms and typically contain very little contextual information [3].

Its main idea consists in choosing important terms from previously retrieved documents that have been identified as relevant by the user and using them to get a new, better query. An analogous process can be done with the documents identified as nonrelevant, deemphasizing those terms in any future query reformulation. The goal of this process is to *approach* the query to the relevant documents and to *move away* from the nonrelevant ones.

This can be illustrated in the following algorithm:

- Identify the N top-ranked documents (using a rank model)
- Identify all terms from the N top-ranked documents
- Select the feedback terms (with one of the term models)
- Merge the feedback terms with the original query (with formula 1)
- Identify the top-ranked documents for the modified queries (i.e.: start again with the reformulated query)

All this process comes from the analysis of how to obtain the optimal request [6]. We start considering Q_0 the query the user entered. The system then formulates a new query Q_1. This new query should be closer to the optimal query as we already explained. The success of this process will depend on how good the initial query is and on how fast query Q_n converges to the optimal query.

Actually, we are searching an optimization function f:

$$Q_1 = f(Q_0, R, S)$$

[3] http://ir.dcs.gla.ac.uk/terrier/

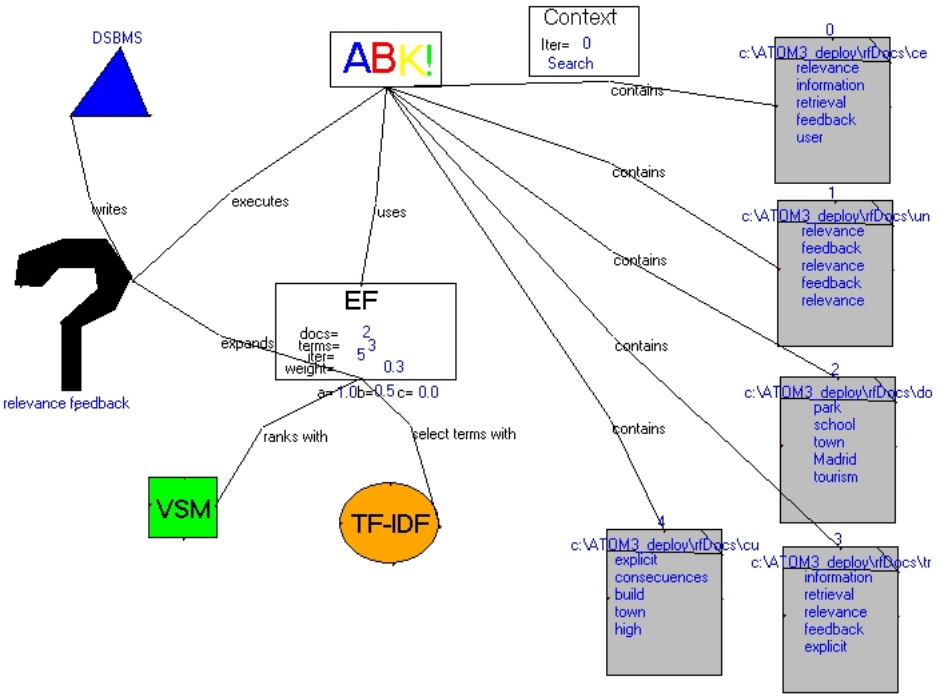

Figure 5: Example used in the simulation

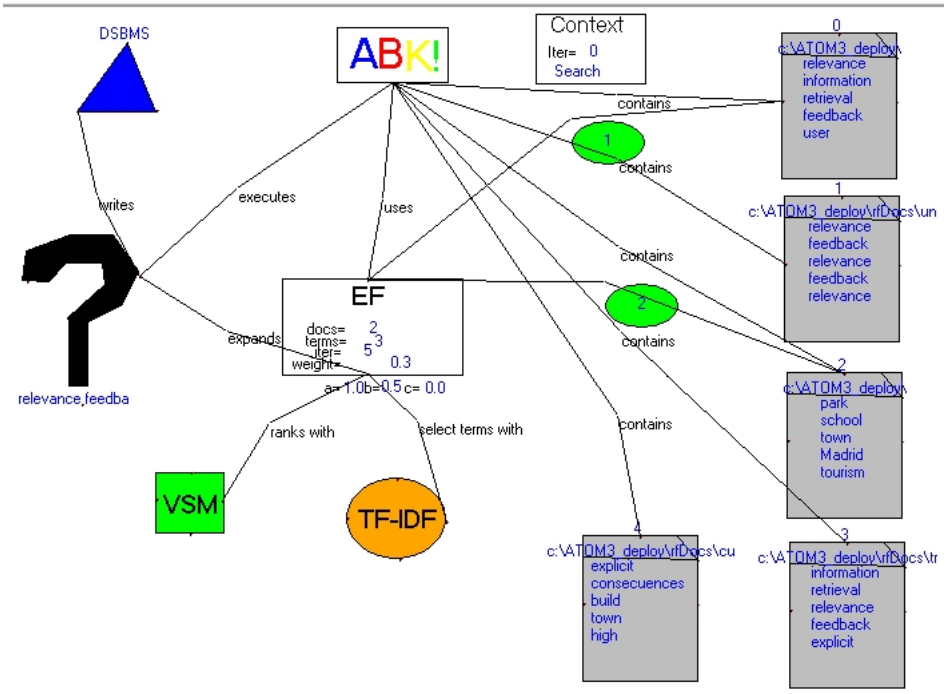

Figure 6: Two documents already ranked (search phase) by the system

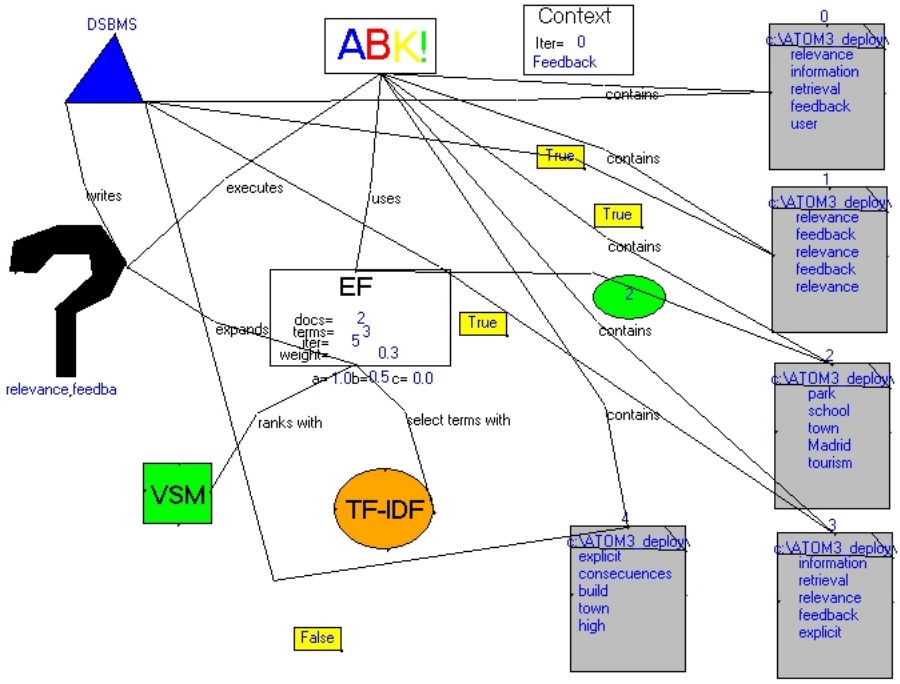

Figure 7: Some documents voted (feedback phase) by the user

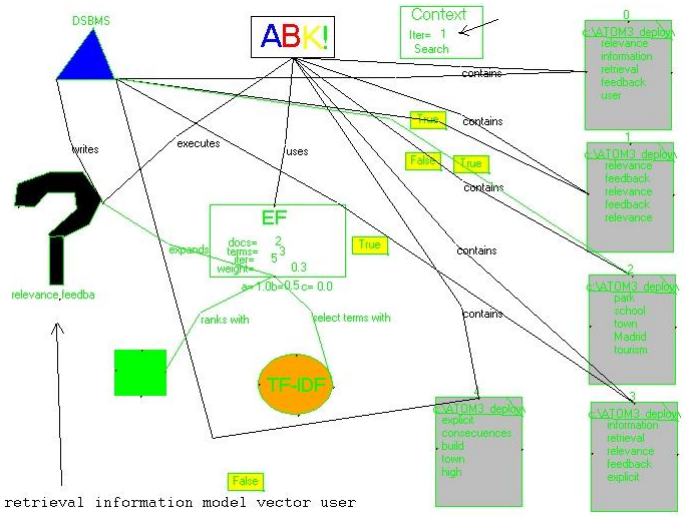

Figure 8: First iteration done, note the expanded query.

where Q_0 is the initial query, R is the subset of the retrieved set that the user thinks are relevant, and S are the documents that the user thinks are nonrelevant within the retrieved documents. The set R should be given a positive sign, because these documents are wanted, for the same reason the set S must receive a negative sign. This suggest the following function:

$$f(q, R, S) = q + \alpha \sum R - \beta \sum S$$

This formula is like the one written by Rocchio in [4]:

$$Q' \quad = \quad a \cdot Q + b \cdot \sum R - c \cdot \sum S \tag{1}$$

where Q is the original query vector, Q' is the new query vector, R the set of relevant document vectors, S the set of nonrelevant ones and a, b, c are constants (Rocchio weights). Typically $a = 1, b + c = 1$, but there are some variations [2]:

- $a = 1, b = 1/|R|, c = 1/|S|$
- $a = b = c = 1$
- Use only first n documents from R and S
- Use only first document of S
- Do not use S ($c = 0$)

There are also some variations in the way the system sorts the relevant terms from each document:

- By the number of documents that contain the term t (n)
- By the number of occurrences of term t (tf)
- By $n \cdot idf$, where idf is the inverse document frequency, obtained by dividing the number of all documents by the number of documents containing the term
- By $tf \cdot idf$

When the system presents the documents to the user it has to rank them, this is a very important decision but the more common model used is the Vector Space Model (VSM), in which the distance (similarity) between two documents is determined by the following formula:

$$cos\theta = \frac{d_1 \cdot d_2}{||d_1||||d_2||}$$

where d_i are document vectors, where each dimension corresponds to a separate term, in the classic form of VSM their coordinates are weights corresponding to the $tf \cdot idf$ factor.

After all these formulae, does relevance feedback have any advantage? Yes, but it also has some inconvenients:

Advantages

- Relevance feedback usually improves average precision by increasing the number of good terms in the query
- It hides to the user the details of the query formulation process
- It is collection independent
- It breaks down the search operation into a sequence of small search steps, designed to approach the wanted subject area gradually

- It provides a controlled query alteration process designed to emphasize some terms and to deemphasize others, as required in particular search environments

Disadvantages
- More computational work
- Easy to decrease effectiveness (one horrible word can undo the good caused by lots of good words)
- It works only on a per query basis, so each time a user issues a new query a new model has to be constructed

Finally, we can classify the relevance feedback process by the *element* who gives feedback:

Explicit feedback Feedback is given by the user.

Implicit feedback Feedback is given by the system after infering the user needs through her previous actions (which documents she views and for how long).

Blind / Pseudo feedback Feedback is given by the system automatically, assuming that the top n documents in the result set are relevant.

We have focused our explanation in the explicit case, because this is the one which involves the user in the feedback and, for this reason, gets results closer to the ones expected by the user.

3 Future work and conclusions

The use of ATOM^3 tool for metamodeling our process has provided us a lot of advantages (such as the possibility of creating a visual language or the described simulation) but it also has made impossible to do, in the time available for this work, some desirables things, such as the followings:

- This work has been centered in the explicit form of the relevance feedback, nevertheless it could be extended to provide the same functionality, for example, simulation, to the other forms (implicit and blind feedback).
- Although we have included in our model some search models (vector space model, language models, ...) reaching their complexity and accuracy is impossible, however this was not the goal of this proposal, but to show in a simple way how the relevance feedback process works and simulate it easily with different parameters.
- It would also be interesting analyzing the properties of some given model or instance, properties such as reachability, boundedness or the existence of single-paths[1]. These properties could be studied with a transformation into Petri nets or similar.
- About the code generation it would be complete if the aforementioned program would be fully developed.

In this work we have modeled in detail, at least theoretically, the process known as relevance feedback. We have also provided a simulation for the special, original case of explicit feedback, in which the user is the one who marks as relevant / not relevant each result

For making all this possible, it has been needed deep comprehension of the process to study (relevance feedback) and of almost every topic in the current course.

References

[1] J. Esparza and M. Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the European Association for Theoretical Computer Science*, 52:245–262, 1994.

[2] David Grossman and Michael Lee. Relevance feedback. Lecture, 2001.

[3] Chris Jordan and Carolyn Watters. Extending the rocchio relevance feedback algorithm to provide contextual retrieval. pages 135–144. 2004.

[4] J. J. Rocchio. *Relevance Feedback in Information Retrieval*, chapter 14. The SMART Retrieval System–Experiments in Automatic Document Processing. Prentice Hall, Englewood Cliffs, NJ, 1971.

[5] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. pages 355–364, 1997.

[6] Mark van Uden. Rocchio: Relevance feedback in learning classification algorithms.