Gesper: Support to Capitalize on Experience in a Network of SMEs*

Maura Cerioli[1], Giovanni Lagorio[1], Enrico Morten[2] and Gianna Reggio[1]

[1] DISI–Dipartimento di Informatica e Scienze dell'Informazione,
Università di Genova, Via Dodecaneso 35, 16146 Genova, Italy
e-mail: {cerioli,lagorio,reggio}@disi.unige.it
[2] Softeco Sismat S.p.A.
WTC Tower - Via De Marini 1, 16149 Genova, Italy
e-mail: enrico.morten@softeco.it

Abstract. Small and medium enterprises (SMEs) are the most affected by the exponentially increasing complexity of the average software system: not losing the grip on the new technologies may turn out to be an unsustainable drain on productive effort, as their developers need to devote a substantial part of their time to learning instead of producing. In order to support continuous education and gathering/reusing solutions, SMEs need a tool supported management of experience and knowledge, providing problem-driven searches.
We describe the experience gained in designing and developing *Gesper*, a tool for knowledge sharing that provides semantic searches based on ontologies. This tool has been tailored to the needs of SMEs and a prototype implementation has been built using free and open source tools.

Introduction

The complexity of the average software system is increasing each year, making production and maintenance more and more difficult. Indeed, such a fast growth may be sustained only thanks to the constantly improved technologies and methodologies for software development. Therefore, it is mandatory for software producers to keep their personnel and processes always updated.

Unfortunately, not losing the grip on the new technologies may turn out to be an unsustainable drain on the productive effort, requiring developers to devote a substantial part of their time to learning instead of producing. This is particularly true for small and medium enterprises (SMEs in the following), where the (human) resources are scant and the budget small.

Thus, many SMEs give up on a well planned effort. But, in order to keep in contact with the evolution of tools, methodologies and technologies, they rely on a plethora of ill assorted knowledge gathering efforts, like individual learning initiative, endeavors by team members to somehow cope with the new tools and

* This research has been supported by the Parco Scientifico e Tecnologico della Liguria s.c.p.a. - POS. N. 23 Avv. 2/2006.

technologies required by challenging projects, and acquisitions of new personnel, with a different cultural background. Then, the bits and pieces of knowledge so randomly acquired and the information about the availability of reusable assets produced by a team are expected to spontaneously spread through the personnel. But, relying on this person-to-person schema of knowledge dissemination has two main flaws. First of all, the person having the crucial bit of information for one project may be currently tied up into another one and hence not assignable to the lacking team. Thus, the knowledge cannot be accessed and shared at the required moment, even if it is available inside the SME. Moreover, the availability of the expertise (reusable resource) may be unknown to the people needing it, because they are not sufficiently familiar with the expert (the team that produced it). The obvious solution to these problems is to have a more formalized management of experience, knowledge, and resources available within the SME[3] allowing the needed information to be found without having to rely on the social network. However, any formal process has its costs and to keep them as low as possible, the best solution is to support the required activities by a tool. Though some approaches and software systems support knowledge management and collaborative working (see e.g., [3,14]), none of them is satisfactory for the management of the enterprise knowledge and experience in a SME. Indeed, in most cases those systems are specialized for a restricted kind of experience or documents in a particular phase of the development, like, for instance, the tools for code management (see, e.g., *Eclipse* [7]). The more general systems are usually expensive and complex, being designed for the application in large companies, and they require a difficult and time-consuming tailoring. Moreover, quite often such tools are capable of supporting only a specific process model and are difficult, or altogether impossible, to adapt to those in use in a given SME. To the best of our knowledge, ViSEK [8] is the tool whose aim is closest to ours. However, ViSEK is a wide-spectrum portal targeting a public sharing of knowledge and resources on software engineering. Instead, we need to model a more sophisticated mechanism, where resources are owned by individual SMEs, which can decide to keep some of them private, share some others with their partners, and even publish a few, possibly in a restricted version. Therefore, a fine-grained visibility policy has to be provided. Thus, the production of a new system appears to be the only viable solution to the problem. In this paper we describe a prototype of such a system, called *Gesper*, from GEStione dell'ESPERienza (Experience Management), which is currently under development by a group of SMEs and the Department of Computer Science (DISI) of the University of Genova.

The requirements and the design of the resources managed by Gesper are discussed in Section 1. Section 2 is devoted to the architecture and a sketch of the implementation. At the end we briefly discuss our results and further work.

[3] Public information can quite easily found on the internet by using tools like Google. The challenging problem is how to search data, which the SME wants to keep private.

1 Experience Management in a Network of SMEs

Designing a tool for managing experiences and knowledge for a cluster of cooperating SMEs has been more challenging from the viewpoint of its integration in the productive process and policies than technically. In the following subsections we discuss Gesper requirements and the part of design that is implied by them.

1.1 High-level Requirements of Gesper

In order to help the SMEs to keep up with the ever-evolving challenges of software production, we have identified a few strategic points to be tackled.

First of all, the tool shall manage *easy to access* information, resources and experiences, making them immediately expendable assets of the enterprise. Notice that experiences, being memorized in the system (as opposed to inside the head of the person who actually made them) will be available independently of the career choices of the experience originators. Ideally, people will access the system to solve a specific problem and will get the correct resources, which could be references to experts, hints or carefully documented explanations of the solution(s) found by other people in previous projects, tutorials, web references and so forth. In Subsection 1.2 we will illustrate the conceptual model of managed resources. It is worth noting, however, that the main requirements are parametric on this model, being more concerned with searches and resource lifecycle, which are independent from the structure of the managed resources. Thus, most part of the requirement analysis and design of the tool discussed here, also applies to other contextes where the resources are differently modelled.

As we argued, the resources to be hoarded and searched should have not only a structure making immediate to extract knowledge from them, but also a clear connection to the problems they refer to, so that the search can be problem-driven, avoiding most false positives. To support this *semantic* search, resources should be indexed using a common dictionary including the most relevant terms of the problem domain. Thus, technically we need an ontology describing the problem domain, so that the resources may be indexed by keywords chosen from this ontology. Notice that the ontology is a configuration parameter of the system and, though Gesper is based on an ontology for software development, discussed in Subsection 2.1, it could be adapted to a different application domain to support the management of experience for another market.

A second important point is the integration of the managed resources into the company organization. This, on the one hand, means that they need to have a well-defined lifecycle. That is, newly inserted resources have to get an approval in order to become visible to users other than who has inserted them. In this way, for instance, a practitioner writes the first draft and submits it while a person more expert in company policies checks that the form agrees with the documentation standards and that security concerns about private data disclosures are met. This two-steps process also helps junior or newly hired staff to learn the company practices by comparing their original draft to the published version. Moreover, the resource will be updated or declared obsolete by an analogous review process,

so that the repository will not be cluttered by useless information. On the other hand, the integration requires the resources to have different levels of visibility, so that the results of a query will depend on who is performing it, possibly granting more privileges to employees of the SME owning the system than to those working for partner companies, and limiting guest access to public data.

Finally, the efforts to document the development process required, for instance, by a quality plan should not be duplicated in order to capitalize on the experience gained during the development. Therefore, the information that can be retrieved from project documents, people curricula, etc should be automatically acquired, pending the check and approval of a human supervisor, of course. Even in the cases when structured knowledge cannot be (semi)automatically extracted from existing documents, it may still be convenient to integrate them in Gesper, to make them searchable in a problem-driven fashion and have their lifecycle managed by the tool. Indeed, such documents may provide a solution, albeit indirect, for some problem missing better answers by structured resources, and they have almost no extra production costs.

So far, we have discussed the requirements of Gesper from the viewpoint of an individual SME. However, the tool should also address the needs of a *cluster* of SMEs, supporting their cooperation. To this aim, the main requirement is the capability of sharing resources in a limited and controlled form, providing different levels of visibility, in order to let each company have full control of its own resources. Gesper is actually designed as a *federation* of instances. Thus, each company in the cluster owns an instance, installed on some local server, containing its own resources. On such server, internal and external users may perform queries with the assigned privileges. The cluster also owns an instance, which is populated with the resources, suitably restricted, of all the individual SMEs, in order to provide a centralized search point. To guarantee that no unwanted disclosures take place, the instance of each company exports a restricted view of its resources to the global instance.

1.2 Conceptual Model of Experience

A central point of the system design is the choice of which resources will be managed by the system, that is, which kinds of resource will possibly be the results of problem-driven searches. Indeed, modelling this aspect captures one of the tailoring needs for the system to be usable by a specific SME.

Figure 1 shows the fragment of UML [16] class diagram describing the resources at a high abstraction level. The root of the conceptual model is the class Resource, representing any kind of managed resource. All the results of a search in the system will be of a type extending Resource. Its attributes and relationships, hence, are those needed by the system to represent resources with a realistic lifecycle in the context of a cluster of companies and to use them for problem-driven searches. Indeed, resources are *tagged* by keywords described by the ontology, so that the *semantic* search is possible. The results of a search are briefly presented by just showing their name and a short description, for the user to decide if a closer examination is worthwhile. Depending on the role of

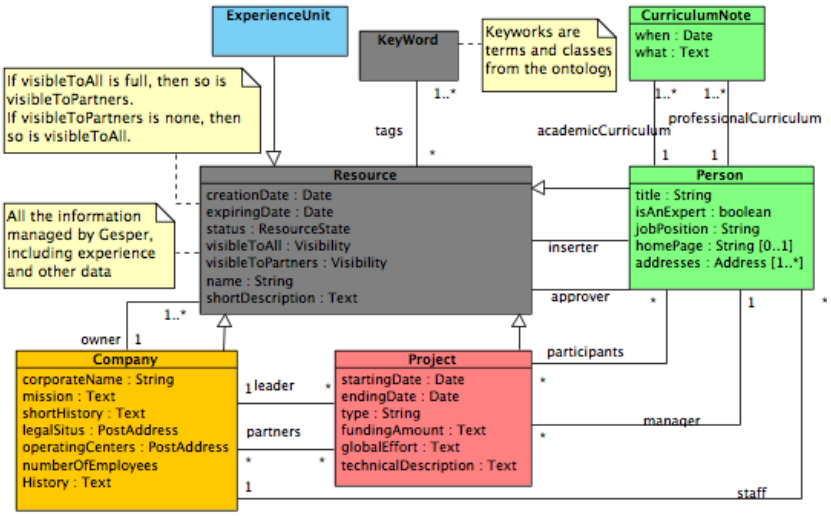

Fig. 1. Conceptual model of resources

the user, who can be an employee of the company owning the resource, or of a partner company, or just any guest, the level of visibility of the resource can be full (that is, the user can see all the details), or restricted (that is, only the name and the short description are shown), or none at all, corresponding to an exclusion of that resource from the search results. For the owning company, the visibility level is always full. However, not everybody in the company staff has the complete control of the resource. In particular, two (possibly coinciding) roles are identified for those entitled to editing: the employee who has submitted the draft of the resource and the manager for approval of the resource. Indeed, each resource has a status, which can be *draft*, *automatically inserted* (hence more prone to contain errors), *approved*, and *obsolete*. The last value is automatically set by the system when the resource has passed its expiring date and can be reset by the approver, who is responsible for all aspects of the resource lifecycle.

Resources are specialized in order to capture different kinds of elements somehow representing knowledge.

The class Person has a flag for distinguishing experts, who can appear as results of a query if their expertise matches the search parameters. Persons who are not flagged as experts may still be represented within Gesper, because they are related to some other resource, for instance they work for the company or are involved in some project, yet, they will not be yielded as results of any query. It is interesting to note that during the requirement elicitation the SMEs involved as clients refused to have the keywords and the expert flag automatically extracted from the curricula of a person. Indeed, our clients want to keep the full control on who should be contacted as expert on a given topic. Thus, Gesper allows an employee to have a deep knowledge of some area, as recorded in his/her

curriculum, and yet not appear as an expert of that topic so that (s)he will not be bothered with requests for help from his/her colleagues.

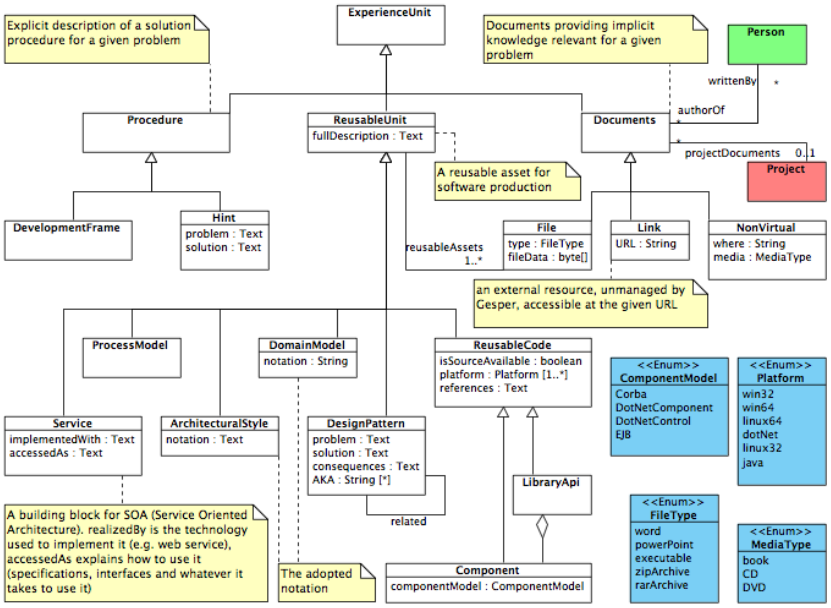

Fig. 2. Conceptual model of resources representing experience

The class Project classifies not only the projects that involves, or involved, the SME owning this installation of Gesper, but also those of partner companies and those which some known expert participated in. Projects are valuable resources, for instance, for solving problems in the area of locating prospective partners in future projects or support of people experienced in the bureaucracy of a specific form of funding. Moreover, projects are a natural source of implicit knowledge in the form of the produced documents, which can be managed as elements of a special subclass of ExperienceUnit.

The class Company has as elements the SME owning this installation of Gesper, owning all the resources inserted by its employees, and the other companies in the SME cluster, the partners in some project, and so on.

The most interesting specialization of Resource is the class ExperienceUnit, which captures the knowledge, both in implicit and explicit forms, accumulated through the experience of the SME, its employees and partners. In Figure 2 we detail the descendants of ExperienceUnit.

The first level of specialization distinguishes the experience units in categories, accordingly to how the user can put the experience to practical use. Indeed, we have Procedure, representing resources that directly describe a procedure to solve a problem, like, for instance, hints and solution for specific problems formalized as *problem frames* (see, e.g., [4]), ReusableUnit, whose instances are

reusable assets to be directly imported and used in the software development, like, for instance, models, code or design patterns [9], and Document, which indirectly provide knowledge, like, for instance, files managed by Gesper itself, or external files, or even physical documents, e.g. books, CDs and DVDs.

The deeper levels of specializations have been only partially worked out and should be extended in a commercial tool. While a large part of the specialization tree having Resource as root could be reused in the model of another system of experience management for SME in different areas, the design of the categories of reusable units and procedures is mostly specific of SMEs in the ICT field and should be reworked for a different application.

1.3 Main Usage Scenarios

Let us briefly describe the usage scenarios concerning the core business of Gesper. As they are quite intuitive, we here summarize them mostly in natural language. However, they were all fully developed as use cases and we propose one of them in full details to let the reader get the gist of Gesper documentation.

When appropriate, the use cases are complemented by sketches of the corresponding GUI, to give the clients a better understanding of the expected interactions with the system.

In general, the user interface shall be as simple and intuitive as possible. It will show a graphical view and a tree representation of the ontology, to allow selecting keywords for resource searching and indexing, together with a research panel and, possibly, several editing/creating resource forms in separate tabs. The capability of inserting and/or editing multiple resources at the same time will be

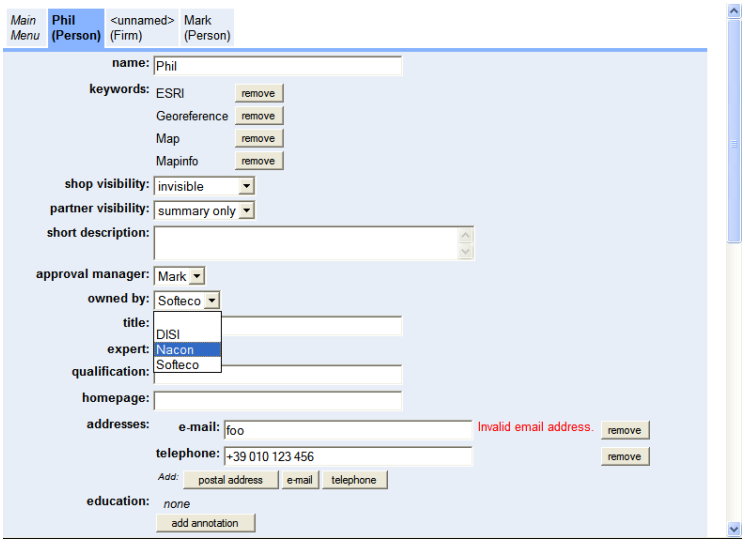

Fig. 3. Screenshot of the *Wizard*

extremely useful when, while editing a resource, say the person *Phil* (as in the example shown in the screenshot in Figure 3), the user notices that the resource should be linked to another that has not been entered yet. Suppose, for instance, that the resource the user is inserting is *owned by ACME*, a firm which is not in the repository. Since the firm is not yet present, its name does not show up in the combo box corresponding to the relation *owned by*. Thanks to the tab-based interface, the user can open another tab to create the new firm, say *ACME*, and save it in the repository. Because the combo boxes are automatically updated, the user can then go back to editing the resource he/she was inserting, choose *ACME* from the combo box and seamlessly continue his/her work.

Resource insertion use case

Name Resource insertion
Primary actors Internal User (IU)
Supporting actors None
Description IU adds a new resource to the system
Triggers None
Pre-conditions IU is logged in the system
Normal flow *Course of actions*

1. IU chooses to create a new resource (specifying its kind).
2. Gesper creates an empty edit tab for the new resource.
3. IU correctly fills in the fields with the resource data:
   - selecting keywords from the ontology using indifferently the graphic view or the tree representation
   - choosing values from lists for enumeration types and resources.
4. IU saves the resource
5. Gesper automatically provides the values *draft* for the *status*, *IU* for the *inserter*, and the *current date* for the *creation date*; then it saves the new resource in the repository.

Post-conditions The repository contains the new resource
Additional requirements None
Notes and issues Data editing and creation will be supported by the GUI using:

- Combo boxes for fields admitting a finite (and reasonably small) set of values (see, e.g., the visibility tags in Figure 3).
- Combo boxes for enumerating resources that could be linked with the one the user is inserting or editing. Furthermore, these combo boxes will be views on the repository, automatically updated when something changes (see, e.g., the `owned by` field in Figure 3).
- Instant field validation, warning the users by means of not obtrusive hints and forbidding to save inconsistent data (see, e.g., near the `e-mail` field in Figure 3).
- The graphical and tree representations of the ontology for selecting the keywords for the resource (not present in the fragment in Figure 3).

Automatic insertion of a documental resource Logged users can choose to insert a document using the *Automatic-Input*, to have the ontology keywords automatically extracted by the tool.

Users select a document file in their (local) system specifying the template[4] it adheres to; for instance, that for *meeting minutes*. Then, the tool parses the document, searching for ontology keywords to be associated with the resource. Gesper completes the other data automatically as for manual insertion and saves the new resource with state *automatically inserted*.

Edit and approval of a resource Managers can review, edit and approve resources. Gesper provides, to logged managers, a list of draft resources to be reviewed. When one of these resource is selected by a manager, Gesper opens an edit tab where the resource can be edited and, optionally, approved.

Searching From the search page, users can insert a search string, choose which kinds of resources they are interested in, and start the search. Gesper shows the list of the matching resources and allows to refine the search string manually or using input from either (or both) representations of the ontology.

When the search string contains one keyword from the ontology, the graphic view of the ontology is centered on that (and the same one is selected on the tree representation). Otherwise, if more keywords are present, then the user is asked to choose the one to use as centering point of the ontology representations.

2 Gesper Architecture and Implementation

In this section we sketches Gesper architecture and briefly discuss the most interesting implementation details.

It is interesting to note that a large part of the architecture has been almost completely fixed by the decision, at the level of requirements, of using already available open-source and freeware software. Thus, the design phase has focused mainly on the definition of the ontology, discussed in Subsection 2.1.

2.1 Ontology

The choice of the ontology is an important part of the customization of Gesper. Indeed, the ontology plays the role of dictionary for the keywords used to tag the resources and hence to direct the searches. Thus, changing the ontology effectively changes the applicative domain (within the limits imposed by the modelled resources).

An acceptable ontology for Gesper should encompass not only the concepts used in software development, covering both the technologies and the processes, but also those specific of the applicative domains targeted by the SMEs using

[4] Gesper has to be customized for the templates defined in a particular SME, for instance by its QA plan. In the prototype we used templates of one of the client SMEs.

the system. Clearly, if all the aspects are fully developed, the ontology risks growing too much, becoming unmanageable. Therefore, we needed a carefully designed ontology that, on the one hand, spreads on different domains, from software engineering to, say, national healthcare organizations and industrial automation, and on the other hand, cut down to only those terms actually used by the interested SME. For instance, cutting down the software engineering terminology to those development processes actually used. Thus, we did not find a satisfactory ontology among the *plethora* of those somehow related to software development (see e.g., [1,15,13,20,19] and [5] for further references), as they were missing the domain specific parts. Moreover, we were not able to combine several ontologies on different domains to get what we needed, as merging the interesting ontologies would have produced a too large result and required careful work to avoid duplicates of concepts on the overlapping areas.

Therefore, we designed the ontology provided with the prototype in a collaborative way with the help of software developers working in several different SMEs, in particular those playing the role of Gesper clients. The current version is still incomplete, with only the major areas fully developed and should be improved in a realistic tool.

The design process of the ontology consisted of two steps.

First we discussed the categories with the aid of a visual model of the ontology, realized in UML, describing the infrastructure of the ontology. The model captures the classes used to categorize the concepts and their relationships, including subtyping, in an immediate way, very easy to understand. Thus, it provided an invaluable support to the several brainstorming meetings needed to finalize its structure. By design, all classes of the ontology contain either individual elements, that is, are leaves of the hierarchy, or only subclasses. In this phase, we also decided which part of the ontology to detail and use to index the resources in the prototype.

The second step of the process was the selection of the individual instances of the leaf classes and the links among those individuals, that is the instances of the relations between classes. This further analysis has been conducted in an asynchronous and concurrent way, starting from a textual file generated from the model and containing the structure of the ontology, that is the list of the classes and their relationships. Then each group involved in the design added to its own copy of the file the interesting instances and their links in fixed format and positions, so that the result could be automatically processed. The resulting files were merged and the complete file was used, on the one hand, to automatically import the ontology into Protégé (see, e.g., [18]), by a Protégé plug-in we have developed for this purpose, and, on the other hand, to update the UML model, by adding the class instances to it.

Finally, the Protégé project was exported in OWL [17] format to be used as a configuration parameter of Gesper. While in the current prototype the ontology is a fixed resource of the tool, an industrial strong tool should allow users to update and extend the ontology as needed. This aim could be easily achieved, for instance, by integrating the Protégé editor within Gesper.

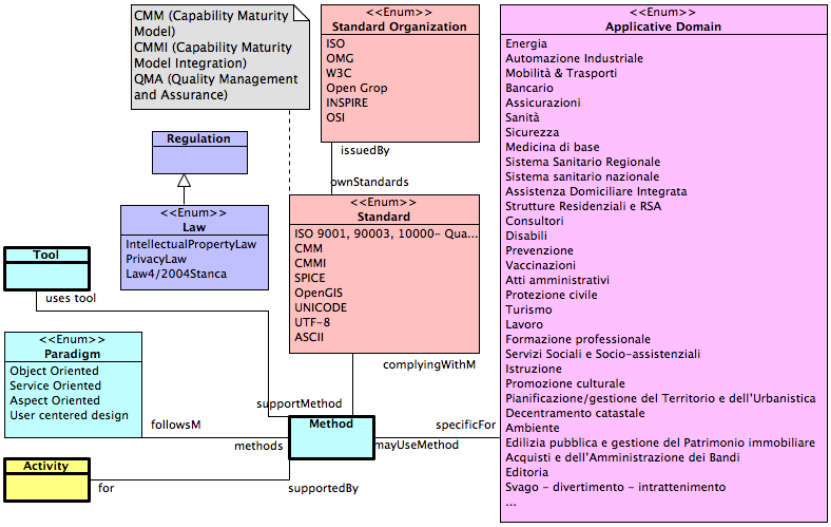

Fig. 4. Ontology: the Root View

The root level of the ontology is depicted in Figure 4. Notice that classes at this level describe the standard concepts of any software development process and are quite stable in time. On the contrary, the elements of the class `Applicative Domain` have been produced by analyzing the current projects of the involved SMEs. Thus, they are specific of the prototype and should be changed for a different productive district. Moreover, even for the same group of users they will change when the client portfolio of the SMEs evolves.

To give just the intuition of the complexity of the final ontology, Figure 5 depicts the fragment of diagram related to the applicative domain and the GIS.

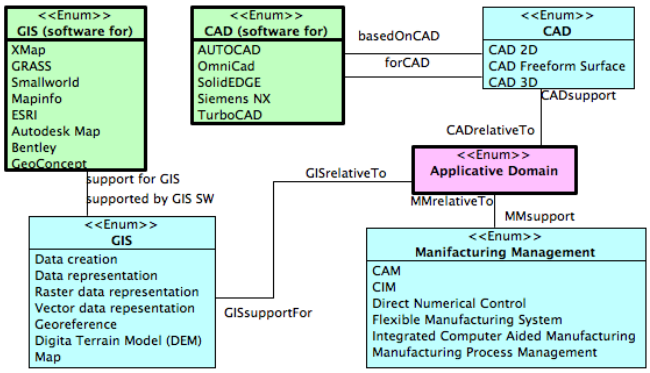

Fig. 5. Ontology: the Applicative Domain View

2.2 Gesper Architecture

The architecture of Gesper consists of several independent modules, detailed below, that work on the *content repository*, providing services like data input, searches and analysis. Figure 6 contains a schematic view of the whole Gesper architecture. The box on the left, *GUI*, represents a generic client, which, in the current prototype, is a graphical user interface hosted inside a web browser. The big rounded box represents an instance of a *Gesper server*, that is, a single node of a *federation*, owned by a SME, that hosts a resource repository and its services. Finally, the box on the bottom right, *Shop*, represents the central node of the federation, which gathers (summarized) data from all the other nodes of the federation and presents a view of this data to the public.

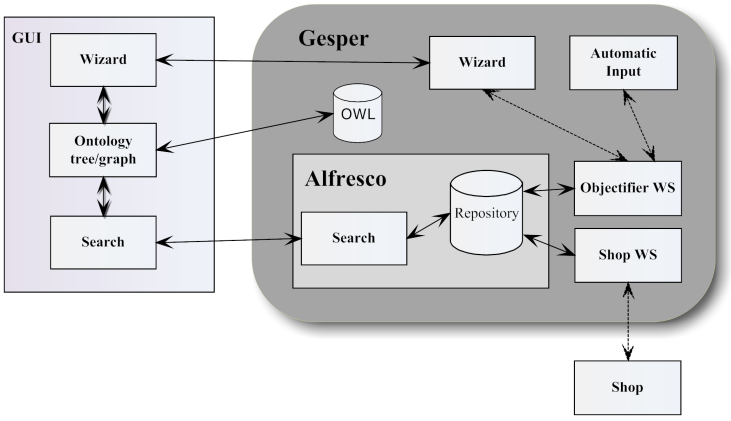

Fig. 6. Gesper architecture

The implementation of Gesper relies on free tools and platforms; at its core we find *Alfresco* [2], a Java-based open source ECM (Enterprise Content Management) providing document management and search facilities. Alfresco offers its functionalities via Web Services and the Java Content Repository API. Moreover, Alfresco includes a web application allowing users to manage the document repository through any web browser. This web application has been customized and enriched to seamlessly work with the Gesper modules interacting with users.

At the moment there are eight modules: *Wizard* and *Automatic-Input* handle data input, *Ontology-Tree* and *Ontology-Graph* allow to navigate the ontology, *Search*, as the name implies, performs searches and *Shop* aggregates some data to showcase the public results and resources of a Gesper federation. Finally, *ObjectifierWS* and *ShopWS* are two auxiliary modules providing simple interfaces to Alfresco; the former maps Java objects to and from the Alfresco repository and the latter allow *Shop* to perform predefined queries.

Some modules run only on clients (for instance, *Ontology-Tree*) or on the server (e.g., *ObjectifierWS*), but most of them consists of a client part, which is a GUI for the users, and a server part, which interacts with the repository.

Let us detail the tasks of each module.

Although both *Wizard* and *Automatic-Input* handle data input, the former presents the users a user-friendly interface to input/edit resources and handle their lifecycles, while the latter performs offline batch acquisitions of (electronic) documents, inferring their metadata by scanning their contents.

Wizard exploits AJAX technology using the *GWT (Google Web Toolkit)* [10] that makes it easier to write high-performance AJAX applications. Using GWT, web applications can be developed in the Java programming language, using full-featured Java integrated development environments, and then compiled into highly optimized JavaScript.

Automatic-Input relies on the Java library *DUO* [6], a wrapper for *UNO* [21], to read Word `.doc` and Open Office `.odt` documents. As mentioned before, *Automatic-Input* needs to read the document contents in order to extract their metadata. The module searches for *keywords*, belonging to the ontology, and other information in fixed, but customizable, positions. For instance, the module can extract the participant list from the minutes of a meeting. This is achieved by using document *templates*, specifying the locations, inside documents, where important metadata can be found and extracted. The prototype handles two templates, one for meeting minutes and the other for development quality plans.

Ontology-Tree and *Ontology-Graph* allow the users to navigate through the ontology. The former presents the entire (linearized) hierarchy of classes and their instances in a tree-based way; the hierarchy has to be linearized because every class can have any number of parents in the ontology. The view offered by *Ontology-Tree* is best suited when the user is an expert, knows the relations between the concepts, and, basically, knows "what to look for" (that is, where are the exact keywords (s)he is looking for in the logical hierarchy). In these cases the tree view offers the quickest access to any part of the ontology. However, when the user is not an expert or needs advice on what entities are related to what is current looking for, the tree based view is not particularly helpful, because it does not represent any relationship but inheritance: in these cases the *Ontology-Graph* kicks in. The *Ontology-Graph* offers a graph view where nodes represent classes and instances, and edges represent relations among those. The view is initially centered on a particular node and, to avoid cluttering the display, only the selected node (the one the view is centered on) and its *directly related* nodes are shown. Users can navigate the ontology by re-centering the view on another area simply by clicking on the node they want to center the view on. The use of this module is best suited for exploratory queries and for navigating through the ontology, in order to gather ideas. Both ontology modules consists of a server-side part, written in Java, and a client-side part written in Javascript. The server components rely on *Jena* [11] for reading the ontology, stored as an OWL [17] file. *Ontology-Graph* uses *jsViz* [12] to render the graphs.

The *Search* enriches the built-in search capabilities of Alfresco with *semantic* searches, exploiting *Ontology-Tree* and *Ontology-Graph* on the client side, and a small layer of custom code, on top of Alfresco search module, on the server side.

The *Shop* provides the users nicely formatted reports summarizing the public data of all nodes inside a federation. *Shop* uses its server side counterpart, *ShopWS*, to query the Alfresco repository. This latter module has been explicitly designed with information security in mind, so it allows to run only queries that have been previously checked and approved. This avoid any leak of sensitive information that could be, innocently or maliciously, gathered by running generic queries run on the repository.

Conclusions and Further Work

We have presented the insight gained by our experience of design and development of a tool for managing experiences and knowledge in a small network of SMEs. In order to contain the development costs, the use of open-source and free software was required from the very beginning. This choice has greatly influenced the design and the architecture of Gesper.

The most original features of Gesper are, on the one hand, the presentation of knowledge in a problem-driven style supported by a *semantic* search based on an *ad-hoc* ontology and, on the other hand, the management of resources directly representing knowledge, as opposed to documents from which the knowledge may be extracted, as it is in most cases.

Being a prototype, Gesper could be improved on a number of aspects, besides a better implementation of the current features.

First, the role of administrator should be supported by tools to manage the ontology, the users, and the templates to be used in the automatic acquisition of documents. The former two activities are currently performed by functionalities of respectively Protégé and Alfresco, which could be integrated in Gesper, possibly making this improvement not much expensive. The ontology maintanance is prioritary, because of the quick changes of the applicative domain, requiring the users to update constantly the terminology used to tag the resources.

A second aspect needing improvement is the management of the Gesper federation and, in particular, how to avoid resource duplications. Indeed, currently if two SMEs share a resource, for instance a document of a joint project, two totally independent instances of the resource exist, in the content management system of each company. Though a better approach to sharing is obviously needed (and solutions could be probably borrowed from the peer-to-peer field), privacy policies and dynamic aspects (for instance employees changing the company they work for) make modelling the federative aspects quite challenging.

An extremely challenging add-in to Gesper, would be to automatically extract knowledge from experience while it takes place, by means of a small wizard helping the users to take notes of the relevant points during their normal workflow, or of watchers, able to record the principal steps taken during procedures, or to compute the resources used for some activity, to help future estimates.

Last, but not least, we need to analyze the user feedback to understand the acceptance of the tool and see how to prioritize its improvements.

References

1. Félix Garcìa, Manuel F. Bertoa, Coral Calero, Antonio Vallecillo, Francisco Ruiz, Mario Piattini, and Marcela Genero. Towards a consistent terminology for software measurement. *Information & Software Technology*, 48(8):631–644, 2006.
2. Alfresco. `http://www.alfresco.com/`.
3. V. Basili and F. McGarry. The Experience Factory: How to Build and Run One. In *Proceedings of the 1997 (19th) International Conference on Software Engineering*. IEEE publishing, 1997.
4. C. Choppy and G. Reggio. A UML-based approach for problem frame oriented software development. *Information and Software Technology*, 47(14):929–954, 2005.
5. Calero Coral, Ruiz Francisco, and Piattini Mario. *Ontologies for Software Engineering and Software Technology*. Springer-Verlag, Berlin, Heidelberg, 2006.
6. OpenOffice.org UNO/Java wrapper. `http://sourceforge.net/projects/duo-wrapper`.
7. Eclipse - an open development platform. `http://www.eclipse.org/`.
8. Raimund L. Feldmann and Markus Pizka. An on-line software engineering repository for germany's SME - an experience report. In Scott Henninger and Frank Maurer, editors, *LSO*, volume 2640 of *Lecture Notes in Computer Science*, pages 34–43. Springer, 2002.
9. Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
10. Google Web Toolkit. `http://code.google.com/webtoolkit/`.
11. Jena – a semantic web framework for Java. `http://jena.sourceforge.net/`.
12. jsViz. `http://code.google.com/p/jsviz/`.
13. OpenCyc. `http://www.opencyc.org/`.
14. Gihan Kim, Minkwang Lee, Jongphil Lee, and Kyungwhan Lee. Design of SPICE experience factory model for accumulation and utilization of process assessment experience. In *Proceedings of Third ACIS International Conference on Software Engineering Research, Management and Applications*. IEEE publishing, 2005.
15. Olavo Mendes and Alain Abran. Issues in the development of an ontology for a emerging engineering discipline. In William C. Chu, Natalia Juristo Juzgado, and W. Eric Wong, editors, *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'2005), Taipei, Taiwan, Republic of China, July 14-16, 2005*, pages 139–144, 2005.
16. OMG. UML superstructure specification v. 2.1.2, 2007. `http://www.omg.org/spec/UML/2.1.2/`.
17. OWL Web Ontology Language. `http://www.w3.org/TR/owl-features/`.
18. Protégé. `http://protege.stanford.edu/`.
19. Sue sen Lin, Fong hao Liu, and Shion fu Loe. Building a knowledge base of IEEE/EAI 12207 and CMMI with ontology. In *Proceedings of Sixth International Protégé Workshop (Manchester, England, 7–9 July 2003)*, 2003.
20. Naiyana Tansalarak and Kajal T. Claypool. XCM: A component ontology. In *Proceedings of Workshop on Ontologies as Software Engineering Artifacts (OOPSLA)*, 24–28 October 2004.
21. Open Office UNO - Universal Network Objects. `http://udk.openoffice.org/`.