Facet-Based Classification Scheme for Industrial Automation Software Components

Vicente Ferreira de Lucena Jr. [*]
University of Stuttgart
IAS - Institute of Industrial Automation and Software Engineering
Pfaffenwaldring 47
D-70550 Stuttgart - Germany
Tel.: +49 711 685 7299
Fax.: +49 711 685 7302
E-Mail: lucena@ias.uni-stuttgart.de

1 Introduction

One of the most important problems at component based software development is how to manage the already developed components. Some approaches reduce this task to storing components in a repository. Nevertheless, even storing components to reuse them later is not as simple as it may seem. As the developer of components is not necessarily the user of components, we need to offer a way to guarantee the right communication among these two professionals. That means we need to support component developers and component users (system developers) to utilize the same vocabulary and documentation in order to improve the communication between them and, consequently, improve the reusability of developed components.

But how should a component be represented, and how this representation must be organized? This is a very important issue as the effective choice of a component requires a deep and detailed understanding of the component's functionality [Fis87, HM91]. To properly use a component in the way it was intended to, the user must understand performance characteristics, function calls, pathological behavior, and so on [Kot98, CR99]. In other words, for a system developer to know what to expect from a distributed software component, the designer must clearly document the component's interface, behavior, and respective operational conditions. Additionally all this information must be easily accessible.

The ease of acquisition of the necessary information clearly affects the scalability of retrieval tools as component libraries grows. That was the main motivation of creating tools to generate catalogues and automatically organize repositories. The main assumption was that the components (very often misused as objects or classes) were developed by diverse people, that could not make use of a predefined documentation structure [GA90, RK91]. They needed to parse whatever documentation they have in order to find similarities that may cluster components together. Some of these tools may be found at [DK93, MMM94, DF95, Hen95].

More important than the ease of acquiring the information is the precision of this information, and the ways to divulge it properly. We assume to work with complex components, more expensive than conventional software [KAO91, SPH98]. They must be exposed in order to be used many times. This exposure will be better done, if we were able to classify the components properly [OHD+92]. That is why the task of classifying and storing components may be done by an expert, capable to well identify the characteristics of a component before its storage in a common repository. The additional costs of this task, minutes of work, will not greatly influence the total cost of a component [MSC99, MPA99].

In fact three main attributes determine how systems developers can successfully reuse a software component: identifiability, understandability, and confidence [FM92]. We enhance the first attribute through a meaningful, domain specific, classification scheme. The last two attributes may be enhanced through well organized documentation associated with an effective presentation of this documentation.

---

[*] The present work was developed with the support of the CNPq, an entity of the Brazilian Government oriented to the scientific and technological development.

We work with different component technologies used in industrial automation. The range may vary from C++ components to JavaBeans or CORBA components. The differences among these technologies make it impossible to adopt a unique formal model for our components, able to cover them all. On the other hand, the choice of only one technology would dramatically limit the coverage of our concept. That is why our decision went in the direction of proposing a unique meta-model for the information presentation of all involved technologies. Our components' presentation is constituted of a collection of documents capable of describing its functionality and constraints well. We also added a domain specific classification scheme specially designed for industrial automation applications.

In the next section we are going to describe our software development process based on components. Section 3 contains our meta-model specially designed for the industrial automation domain. A description of the information needed and the explanation of the adopted classification scheme is done there. At section 4 we present our selection process. An example of how we support users is also presented. Finally we summarize our work presenting our conclusion and our ideas for the future at section 5.

2 Component Based Software Development

The main objective of component based software development is the implementation of software systems by the assembly of prefabricated software building blocks. Since the beginning of the development process we have to invest in the construction of components, that shall be generalized enough to be easily utilized in future projects. Indeed since the analysis phase the desired system should be thought as composed of software components.

Nevertheless, it is not enough to design software components to assure a great level of reuse in a software development process. The developed component itself should have some properties that encourages and facilitates its reuse. A good component should be easily understood, its interface, adjustment parameters, and usage should be clear. That is not an easy characteristic to achieve, because the specification of a component depends strongly on the description of the desired problem domain. A component should also be useful. That is relative to an application's problem domain. As stated in [JSW92] even if a component is functionally rich, the totality of its functionality may not be useful to all applications where it can be inserted. In that way a good component is not one that can do many different tasks, but one that can exactly do what we need to be done. We need to find a component that best fits to the problems we want to solve. A last characteristic of a good component is its usability. A component only reaches a high level of usability, if the effort to accomplish its integration is low. That means, the way we integrate components with one another should be made clear enough to reduce the time of the adaptation rework on the desired components.

Even if we achieve all these features, we can not assume a large margin of reuse. There are more problems that difficult their reutilization, as for example the search and correct understanding of a component already stored in a repository. Newly mechanisms or tools had to be introduced in the development process trying to guarantee that a useful, functional and (re-) usable component could be found [Bon97, BD99]. In our approach we solve this problem introducing a component publication procedure dedicated for the industrial automation domain. During this publication procedure we guide the components developer. Later on, during the selection of pre-existing components we give support for the user of our components, i.e. the system developer.

Figure 1 summarizes our concept of a software development process based on reusable components. The component developers receive the requirements of a desired component from the systems analysis team, at the very first time a component is created. After the necessary development work, the component is tested and its functionality is verified. The component is now almost ready to be used. The last step is the "publication" process. Before storing it in a repository, we classify it and order some information from the developer that may clarify its usage, potentials, and constraints. The classification information is oriented to the industrial automation domain and the descriptional data should add information about the component's performance, its operating environment, and its resource needs. A tutorial system guides the component developer by giving him suggestions during the classification process and by ordering the necessary information about the component.

The development of systems based on components starts up with the reception of the system requirements. System developers should try to proceed with their analysis based on components. As the analysis is completed, they may proceed with the selection of existing components. The way a system developer should proceed with his analyze in order to obtain requirements oriented to components is researched in our institute under the name "Component Oriented Analysis". The selection process is subdivided in three parts, the search of a functionality, the evaluation of the found components, and the final decision about their use. As a result of the searching of components the developers may receive a list with components that fulfil the functionality they are searching for.

Nevertheless, there is no guarantee that the components really attends their requirements completely . The next step should than be a detailed evaluation of the suggested components. Some of them may not exactly fulfil the developers necessities and constraints. On the other hand, some components may be exactly what they need. Developers should decide lastly, among the components they are sure that work properly, which one may better fit to their requirements.

Some interactions between component developers and system developers are possible during this process. The system developer team can command a rework on a component that did not fulfil the published characteristics. They can also detect a failure in an existing component that was not previously observed. Lastly, they can order a brand new component in case they were unable to find one component that could fulfil their needs. These interactions will guarantee the ongoing development of components and systems with a high quality standard and a small percentage of error.

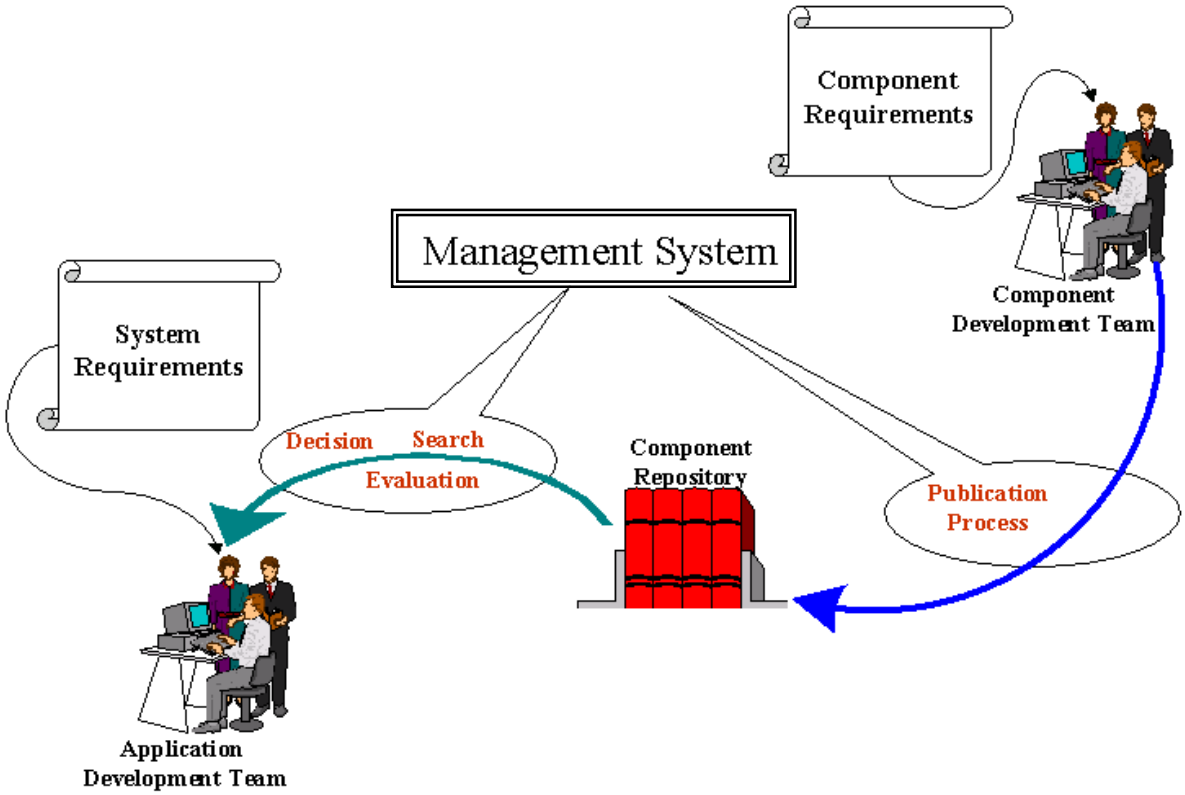

Figure 1 - A simplified software development process based on reusable components.

3 A Meta-Model for Components in the Industrial Automation

As we work with different technologies of components, we decided to specify a meta-model capable of describing all relevant information about them. This model intends to facilitate the search and find of one already developed component, it also supports the comprehension of the component's functionality and usability. Finally, some items have the purpose of enabling a commercial comparison of functional identical components. The component's description is organized in four distinct groups of information. The first group contains general data about the component and an unequivocal component's identification. The second group gather the functional characteristics of the components together. All necessary information for the understanding of the component is stored here. The way a component works, its performance and constraints are clustered in a third group. Finally, the commercial information is arranged together at a fourth group.

These different groups of information were formally represented with an entity-relationship model. Figure 2 shows the resulting diagram. Observe that some elements of the tables are optional and may not be given by component developer. Some items are specific for a particular technology, e.g. the complete documentation of a JavaBeans component will be quite different from the one of an ADA component. In the next paragraphs, special attention will be given to the elements that belong to our classification schema.

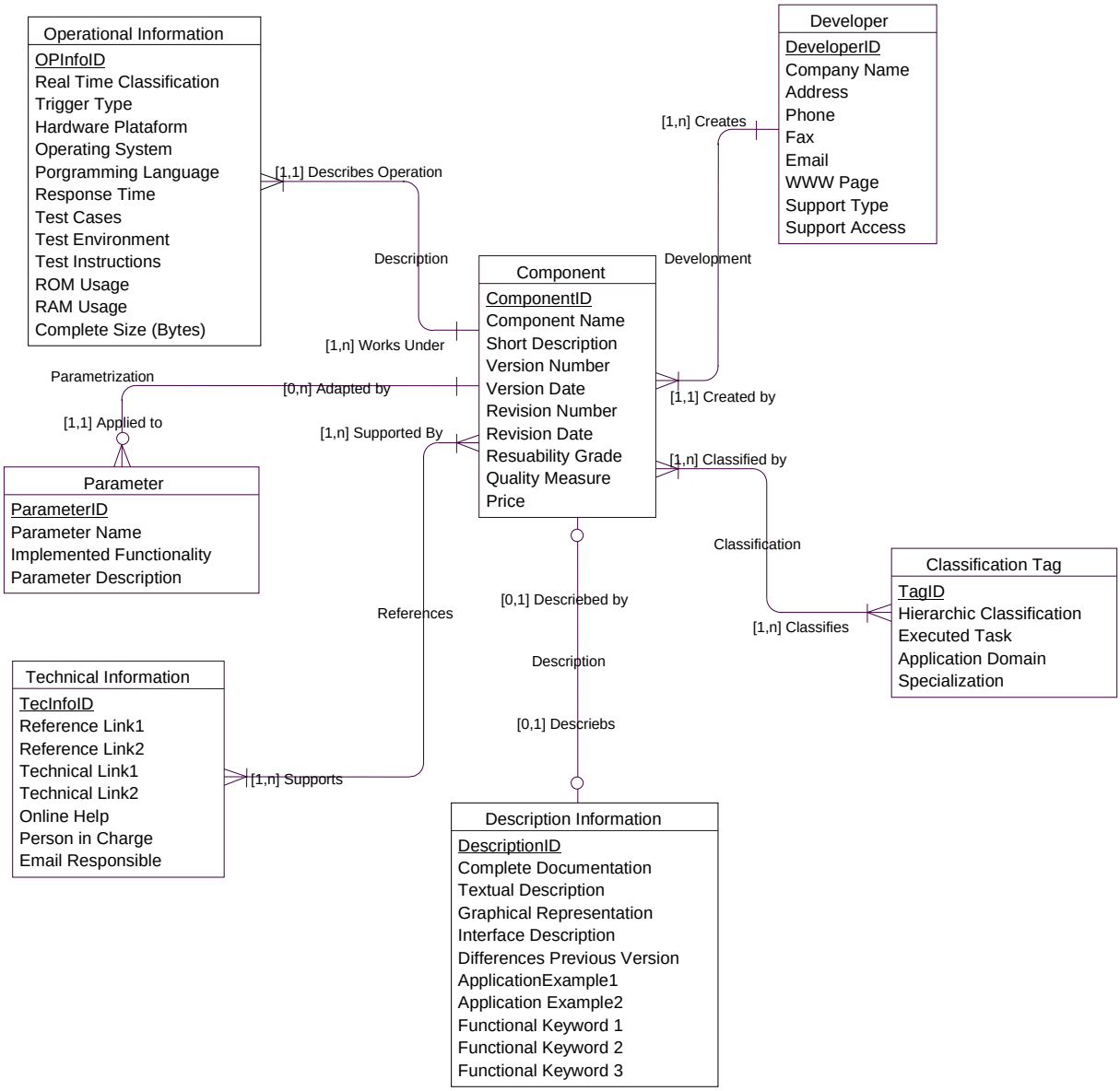

Figure 2 – Entity-relationship representation of our components.

3.1 Facet Classification for Industrial Automation Components

The main contribution for an easy search and find of components is done by our classification scheme. Facet classification has been already applied to many different engineering domains [PD91, MA+97, MMM95]. Common sense is, that the conditions to successfully use facets by the classification of an element are the following:

- a facet must be composed of terms, that meet a meaningful description of the possible users' requirements;
- a facet must be oriented for the reuse;
- a facet must be organized in accordance with the relevance for the user;
- the classified element may have as many facets as need to precisely define its use.

Depending on the terms adopted, faceted classification may enable a very precise classification for each participating element. On the other hand, it can become unclear, if the number of assigned terms to a facet grows. Our approach defines a limited number of facets. Nonetheless, it covers the main aspects and terms of industrial automation applications. Our facet classification scheme is based on a pre-defined hierarchy of terms. This facilitates the orientation of users looking for a suitable term. We made use of the concepts about industrial automation presented at [LG99] to generate the terms. Not all terms, but only the ones relevant for the system developer are filled out. The facets suggested for the classification of components in the industrial automation are described bellow.

a) Application Domain: Industrial automation itself is a large domain. We need to classify our components in a sub-domain in order to determine their possible application. There are two main divisions of the industrial automation [LG99], i.e. product automation and plant automation. Each of them may have many other sub-divisions. That is where our terms are originated. For example, a component belonging to the sub-division "Product Automation" should be classified as "Home Entertainment" another one should belongs to "Medicine Systems". One "Plant Automation" component may receive the classification "Building Systems".

b) Specialization of the Domain: This facet is used to specify the component's area of usage in more detail. The examples mentioned above could receive terms like "Image Adjust" as specialization for "Home Entertainment", or "Fire Protection" as specialization for "Building Systems".

c) Industrial Automation Task: Using standard terms of the industrial automation domain, we express the category of the component's functionality. Among usual terms are: Input device (sensor), Output device (actuator), Supervision, Control, Command, Communication among others. No precise description of the implemented functionality is given, but a high level classification.

d) Hierarchic Classification: Industrial automation applications may be distributed in a hierarchical classification scheme in accordance with their management level. Figure 3 shows the different approaches discussed in [LG99]. They are the levels of hierarchy, levels in distribution and levels of a company. For our facet we adopted the terms Business Level, Production Level, Process Control Level, Process Variable Level and Field Level.

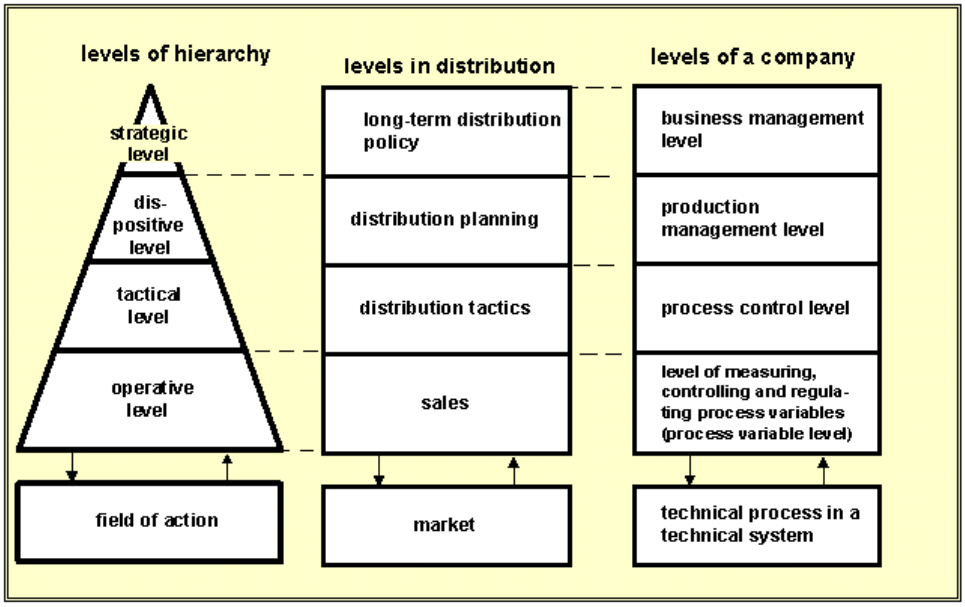

Figure 3 – Levels of management at industrial automation.

e) Implemented Functionality: This facet should precise the implemented functionality of the component. As the goal of the facet is to publish the component's characteristic we need a set of key-words summarizing the components capability. A complete textual description of the functional characteristics of the component is part of the complete model and should be stored later.

f) Trigger Type: Two kind of components are considered, the event-trigged and the time-triggered components. Both types are of great importance at industrial automation applications.

g) Real-Time Characteristic: We adopted three standard values for this facet. They are Hard Real-Time, Soft Real-Time and No Real-Time.

h) Component Technology: The programming language or architecture used may be described by this facet. We may have a functionality implemented using diverse programming languages or architectures. For that case a list with the available technologies and links to the respective components are generated automatically.

i) Hardware Platform: The original hardware system is described by this facet. If a component works with some other platform, it may additionally be inserted.

j) Operating System: All possible operating systems may be inserted here. That is a very important topic at the selection of components.

3.2 Classifying Developed Components

The facets adopted for the taxonomy of our components were chosen based the theory presented at [LG99]. This book is an actualization of a popular text book on industrial automation in south Germany [Lau76]. The classification scheme itself is not enough to assure a well suited classification of our components. Although the terms adopted are relative well known, the hierarchical structure is new and must be divulged in order to guarantee the communication between component developers and component users.

Trying to solve this problem, we conduct component developers during the publication process, the last phase considered at our development process before delivering a component. This conduction is done presenting a well organized on-line help, which is responsible to describe the facets used and to offer taxonomy values for each facet of the scheme. A list with pre-defined values is used as base for the classification of new components. The values are presented dynamically, i.e. if a user decides to classify a component as belonging to the application domain "automobile" (the facet Application Domain will receive the value Automobile), a list with suggestions for the facet Specialization of the Domain will only contain meaningful values in reference to the first choice (e.g. ESP, ABS, etc).

Nonetheless, the responsibility of a meaningful classification of components belongs to the developer, the one who best knows the exact usage of his software component. Developers may adopt one of the suggested values but have also the freedom of introducing new terms whenever they believe these terms should better describe their component. This new value is automatically introduced as a valid one and may be used as a suggestion during the classification of another component. That is an indirect and flexible way of acquiring knowledge from the experts. They may introduce whatever value they want and this value will contribute for the classification of new components. At figure 4 we present a classification example for two components.

	Component X	**Component Y**
a) Application Domain	Product Automation	Process Automation
b) Specialization of the Domain	Medicine System	Assembly System
c) Industrial Automation Task	Pulse Supervision	Linear Control
d) Hierarchic Classification	Field Level	Field Level
e) Implemented Functionality	Pulse Frequency Measure	PID Algorithm
f) Trigger Type	Events-Trigged	Time-Trigged
g) Real-Time Characteristic	Hard Real-Time	Soft Real-Time
h) Component Technology	C++	JavaBeans
i) Hardware Platform	PC Based	Sun Solaris
j) Operating System	Windows NT	Unix

Figure 4: Example for the classification of components.

3.3 Complementary Information about the Components

The facets used by our classification scheme are mandatory, i.e. component developers are only allowed to save their components at the repository after proceeding with a well conformed classification of each one. Some other information contained at our complete description model (figure 2) are optional and may not be divulged. The absence of this data will strongly prejudice the understandability of the components and consequently decrease their probability of being reused. To solve this problem we are planning to develop an advisor system, similar to the one used for the classification scheme, to help component developers to well document their work. This advisor will dynamically adapt itself in accordance with the component's technology and conform our model.

4 Selecting Components Based on the Available Information

The proposed model must help component developers to find, understand and reuse a previously developed component. Not all information contained at the model are obligatory, some items may be included as optional ones. Although all information presented is important for a complete description of the component, its functionality and reuse characteristics, different data may be used in different phases at the search process.

The selection of a component for its reuse is divided into three sub-processes. The first one consists of searching a component. This step will provide system developers with a list of candidates that, in accordance with our previous classification scheme, may be contained at the desired application area. The second step will offer a set of information that may certify, if the presented components met their technical requirements completely. At last, another group of information will permit developers to decide, in accordance with commercial aspects, if a component shall be used at their application or not. The contents of each group of data and the procedures to be done are described at the following paragraphs.

4.1 Searching

The objective of this procedure is to perform the identification of candidates executing a preliminary assessment based on a classification scheme. The information that will be used at this step are: the general classification based on the component's area of application, the classification oriented to their operational characteristics, the classification dedicated to the functionality information, the desired programming language, and lastly the name of the developer.

The selection will be based on the desired application domain name and its respective specialization, the automation task to be fulfilled, and the position of the desired functionality in the automation hierarchy. The described fields are going to be specified by the component users (system developers or domain experts). That means, the persons who best know what they need. Nevertheless, there should always remain the possibility to leave one classification field without value. In this case all available components for this category shall be presented. Within the operational information we are supposed to specify, as far as possible, the desired operating system and hardware platform. Additionally, developers should specify the application's real-time characteristic (hard, soft or no real- time) and the trigger type (events or time trigged). The last criteria should drive the selection through the desired functionality, possible technology, and component's developer name.

Figure 5 presents the mask we offer to the system developers during the search. The amount of available components decreases as we proceed with the selection. A similar mask is offered to the component developer during the classification procedure, before storing the component at the repository. The values presented are dynamically generated and are based on previously stored data.

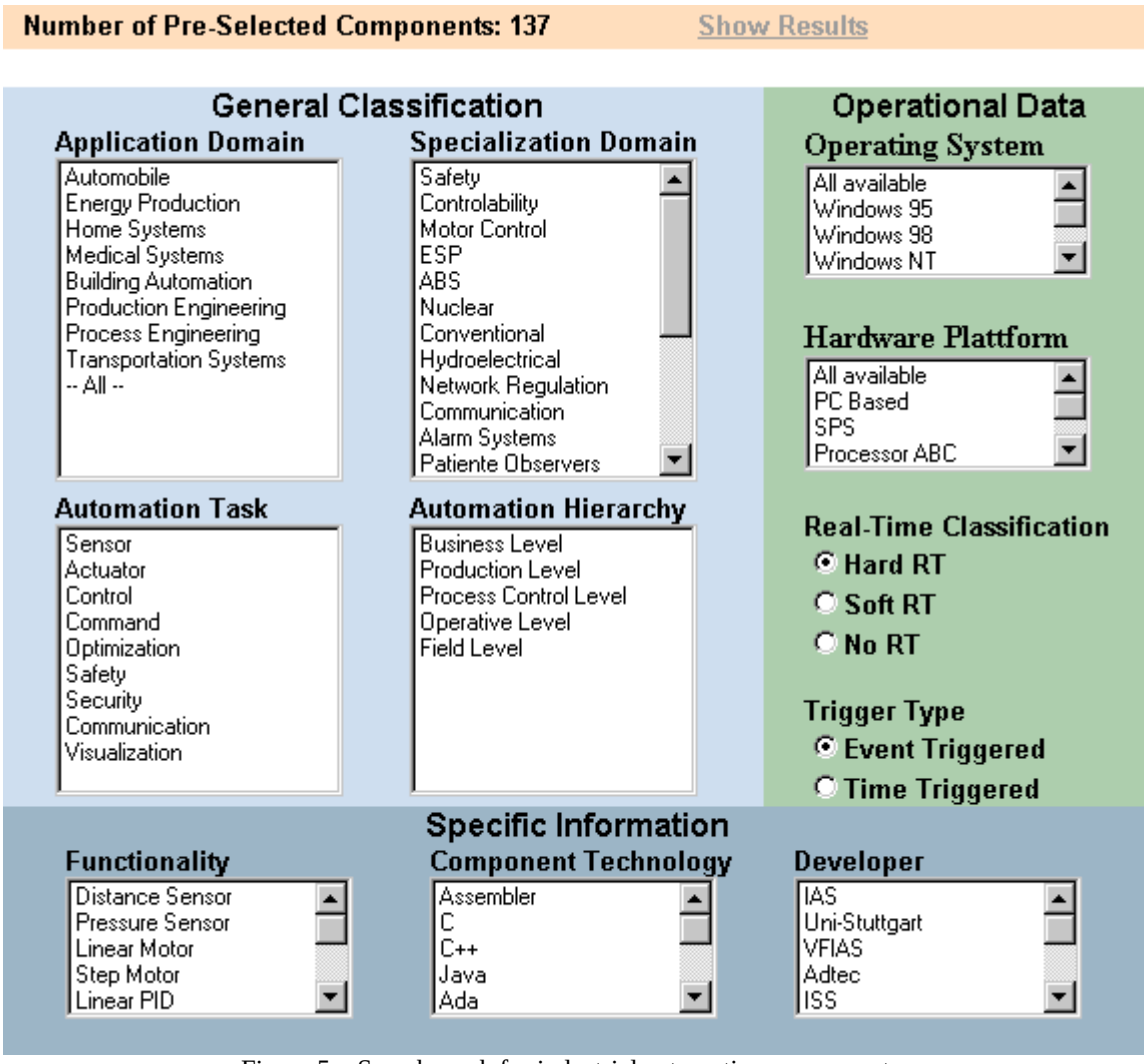

Figure 5 – Search mask for industrial automation components.

4.2 Technical Evaluation

After the selection process, the number of components should be reduced to a small set of potential candidates. The system developers must now verify, if these candidates really meet their needs. The main work here is to test components, either in the current development and/or deployment environment or in closely-related environments, to confirm that each component runs as advertised, and meets the desired requirements.

At this point some support may be given to developers in order to facilitate the evaluation process (e.g. simulations or demo versions). The use of checklists, where the most important original requirements are presented, facilitates a confrontation with the divulged component's characteristics. This may help concluding, if a component was applicable or not. Although the technical evaluation is an excluding step, it is not enough to conclude which component is the right for the developer's business. In fact, more than one component may have the same characteristics and equivalent performance, and shall be eligible for the next evaluation step. The final conclusion must wait until we perform the commercial evaluation.

4.3 Decision Making

Now the system developers may work only with the components that were approved in the previous steps. The goal is to determine which component may be integrated into their system. The commercial evaluation shall compare the pre-selected components enabling system developers to make a final decision. Statically, the amount of possible components that arrive at this step is small. In despite of it, the final evaluation should be executed even if only one component could be pre-selected. Items like the developer marketplace and the price of components may be an exclusion factor.

Another mechanism we have introduced to improve the communication among users of our system is an automatic request generation of a new component. In case a user do not find a desired functionality, he will be assisted to generate an order which is also based on the classification scheme presented.

5 Conclusion

The usage of a large collection of reusable, domain specific, software components introduces new organizational problems at the software development process. The best form how to locate and retrieve appropriate components, and how to meaningfully browse and navigate among similar components is one of these problems. As the one who develops components is not necessary the same who reuse them, we need to offer a way to guarantee the right communication among these two actors. In other words, we need to support component developers and component users to utilize the same vocabulary and documentation in order to improve the reusability of the developed components.

The meta-model described above contains the most important information for the search, evaluation, and reuse of components for industrial automation applications. The central idea was not to propose one formal model. The main reason for our decision is that it would be impossible to develop only one model to cover all technologies used in industrial automation systems at our institute. The data used is enough for a precise selection process. The selection is divided in three sub-processes, the searching of a desired functionality, the technical evaluation of pre-selected candidates, and the final decision based on the information previously obtained plus commercial ones.

We give support to component developers during the classification of their components. Our classification scheme is based on facets specially inspired at industrial automation applications. A dynamic generation of suggestions helps developers to classify their components properly. The terms suggested are in accordance to previous experiences. Additional support is given in order to guarantee that all necessary documentation will also be stored. With that information, system developers may decide on the possible use of a particular component. Lastly, developers may compare the approved components and decide on their usage.

The system is under evaluation in our institute. In a near future we plan to migrate it to an open system in the Internet. The idea is to permit any component developer to publish his components at our system, increasing the reusability of them. New issues are going to integrate the tutorial system. We plan to formulate a quality measure and a reusability grade for each component. These measurement units will enable a quantitative approach for these two important aspects of component reuse.

6 Bibliography

[BD99] Borgida, A.; Devanbu, P.: *Adding more "DL" to IDL: Towards more Knowledgeable Component Inter-Operability.* Proceedings of the 21th International Conference on Software Engineering, 1999, pp. 378 – 387.

[Bon97] Bondeli, P.: *Developing Reusable Multi-Tasking Components Using Object-Oriented Techniques in Ada 95.* Proceedings of the 8th International Workshop on Real-Time Ada, 1997, pp. 33 – 34.

[CR99] Cicalese, C. D. T.; Rotenstreich, S.: *Behavioural Specification of Distributed Software Component Interfaces.* IEEE Computer, July 1999, pp. 46 – 53.

[DF95] Damiani, E.; Fugini, M. G.: *Automatic Thesaurus Construction Supporting Fuzzy Retrieval of Reusable Components.* Proceedings of the 1995 ACM Symposium on Applied Computing, 1995, pp. 542 – 547.

[DK93] Dunn, M. F.; Knight, J. C.: *Automating the Detection of Reusable Parts in Existing Software.* Proceedings of the 15th International Conference on Software Engineering, 1993, pp. 381 – 390.

[Fis87] Fischer, G.: *Cognitive View of Reuse and Redesign.* IEEE Software, July 87.

[FM92] Faget, J.; Morel, J. M.: *The REBOOT Approach to the Concept of a Reusable Component.* 5th Workshop Institutionalizing Software Reuse, 1992.

[GA90] Gossain, S.; Anderson, B.: *An Iterative-Design Model for Reusable Object-Oriented Software.* Proceedings of the European Conference on Object-Oriented Programming Systems, Languages, and Applications, 1990, pp. 12 – 27.

[Hen95] Henninger, S.: *Supporting the Process of Satisfying Information Needs with Reusable Software Libraries: an Empirical Study.* Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, 1995, pp. 267 – 270.

[HM91] Helm, R.; Maarek, Y.S.: *Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries.* Proc. of the OOPSLA 1991, pp. 47 – 61.

[JSW92] Jensen, J. S.; Stewart, H. D. and Whittington, P. H.: *Successful Experience with AdaSAGE Reusable Component Library.* Conference Proceedings on TRI-Ada '92. 1992, Pages 276 - 280.

[KAO91] Krutz, W. K.; Allen, K.; Olivier, D. P.: *The Costs Related to Making Software Reusable: Experience from a Real Project.* Proceedings of the Cconference on Ada: Today's Accomplishments - Tomorrow's Expectations, 1991, pp. 437 – 443.

[Kot98] Kotula, J.: *Using Patterns to Create Component Documentation.* IEEE Software, March/April 1998, pp. 84 – 92.

[Lau76] Lauber, R.: *Prozessautoamtisierung Band 1.* Springer – Verlag Berlin – Heidelberg, 1976.

[LG99] Lauber, R.; Göhner, P.: *Prozessautoamtisierung 1.* Third Edition. Springer – Verlag Berlin – Heidelberg, 1999.

[MA+97] Mili, H.; Ah-Ki, E.; Godin, R.; Mcheick, H.: *Another nail to the coffin of faceted controlled-vocabulary component classification and retrieval*; Proceedings of the 1997 symposium on software reusability (SSR'97), May 1997, Boston USA, pp. 89 -.

[MMM94] Mili, A.; Mili, R.; Mittermeir, R.: *Storing and Retrieving Software Components a Refinement Based System.* Proceedings of the 16th Int. Conference on Software Engineering, 1994, pp. 91 – 100.

[MMM95] Mili, H.; Mili, F.; Mili, A.: *Reusing Software: Issues and Research Directions*; IEEE Transactions on Software Engineering, Vol. 21 No.6, June 1995.

[MPA99] Martín, F. J.; Plaza, E.; Arcos, J. L.: *Knowledge and Experience Reuse Through Communication among Competent (Peer) Agents.* International Journal of Software Engineering and Knowledge Engineering, Vol. 9, No. 3, 1999, pp. 319 – 341.

[MSC99] Molina, M.; Sierra, J. L.; Cuena, J.: *Reusable Knowledge-Based Components fo Building Software Applications: A Knowledge Modelling Approach.* International Journal of Software Engineering and Knowledge Engineering, Vol. 9, No. 3, 1999, pp. 297 – 317.

[OHD+92] Ostertag, E.; Hendler, J.; Díaz, R. P.; Braun, C.: *Computing Similarity in a Reuse Library System: An AI-Based Approach.* ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 3, July 1992, pp. 205 – 228.

[PD91] Prieto-Diaz, R.: *Implementing Faceted Classification for Software Reuse*; Communication of the ACM, Vol. 34, No. 5, May 1991.

[RK91] Rosenblum, D. S.; Krishnamurthy, B.: *An Event-Based Model of Software Configuration Management.* Proceedings of the 3rd International Workshop on Software Configuration Management, 1991, pp. 94 – 97.

[SPH98] Smith, R. K.; Parrish, A.; Hale, J.: *Cost Estimation for Component Based Software Development.* Proceedings of the 36th Annual Conference on Southeast Regional Conference, 1998, pp. 323 – 325.

[Ste98] Sterling Software Application Management Group.: *The CBD96 Standard Version 2.1.* Standards for Specifying and Delivering Software Components Using COOL:Gen; P.No. 261639-004; July 1998.