Integrating search methods on Active Documents

Maria Ester Soares Xavier, Flávio Miguel Varejão

Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, s/n - Campus de Goiabeiras
CEP 29060-900 - Vitória - ES – Brazil
ester@tropical.com.br, fvarejao@inf.ufes.br

Abstract. In this work we present an extension of the parametric network representation language used by Active Documents models. The extension incorporates a new kind of parameter to the representation language. This new kind of parameter enables the reuse of a task-especific library of problem solving components implementing automatic search in Active Documents Systems. In order to illustrate the use of this new parameter, the Hill Climbing, Propose and Backtracking, Complete the Model Then Revise and Extend The Model Then Revise methods were implemented. These methods were used for solving an office allocation problem. The knowledge modeling technique used here allows the knowledge engineer to configure new methods using the library components. The knowledge engineer can also include new components to the library.

1 Introduction

Knowledge Based Systems (KBS) have had success on many types of applications. However, they have trouble solving problems where it is not possible (or it is not easy) to anticipate all necessary knowledge during the knowledge acquisiton phase.

In order to solve this type of problems, some approaches have proposed a partnership between the human user and the computational system. While human users may use their common sense, creativity, learning capability and experience knowledge to guide the search for a solution, the computer system may efficiently examine a large space of alternatives.

Active Design Documents (ADD)[1] are computational systems based on this idea. An ADD system acts like an apprentice who assists the designer performing automatic verifications, calculations, suggesting solutions and recording the design decision process. They are called active documents because they are able to answer questions on demand about the designed artifact and about the whole design process.

[1] [1] Even though ADD systems were originally developed for supporting design tasks, they are also able to support other types of problem solving tasks.Therefore, we just call them active documents.

The main idea of ADD systems consists of making the user responsible for taking the final decisions and guiding the search process. It means that, in ADD systems, the human user defines the states to be tested and changed. Due to this strategy, the user may use his specific abilities to reduce the search space and avoid that the computional system becomes lost within a combinatorial space.

However, in some situations, there may be effective search methods that could be applied for solving a problem. On these cases, the ADD approach may demand an extra effort from the user and, sometimes, limit the quality of results, since the number of alternatives tested by the user is much lesser than the number tested by an automatic search method.

This work proposes an extension of the ADD representation language that enables these systems to perform automatic search on state spaces. This extension is based on an approach of knowledge component reuse proposed by Motta [4]. It allows the knowledge engineer to use different search methods for supporting problem solving. Besides, this extension may enable the ADD system user to choose the most appropriate method to be applied in a specific situation during problem solving.

Section 2 of this paper reviews the active documents approach. Section 3 shows how search methods are integrated to ADD systems. Section 4 presents our conclusions.

2 Active Documents

ADD systems are based on the metaphor of having a computerized apprentice following the user in the problem solving process, recording his rationale and helping him to take decisions. Two general types of interaction may be performed: the user may ask the apprentice to perform some calculations and make decisions; or the user may perform these actions and ask the apprentice to follow him, verifying and recording his decisions.

The ADD system is able to perform these activities due to its decision-making process knowledge. If its knowledge is not appropriate to make a specific decision, the user must change the knowledge base, adjusting the ADD system behavior. The adjusted knowledge base will be used for explaining the decision process rationale (i.e., why the decisions were made on that way).

An ADD system knowledge base is described using a parametric network representation language. In this network, parameters represent problem requirements or decisions that should be made during the decision process. The decisions are performed if the parameter is chosen during the decision process. The archs describe parameters dependencies (if the parameter A depends on parameter B then parameter B must be solved before parameter A).

Figure 1 shows a parametric network example used by an ADD system that helps a hypothetical refrigerator system designer. The network arches indicate that it is necessary to know the desired temperature and the volume to be refrigerated for calculating the refrigeration type.

According to Garcia et al. [2], a parametric network may have three types of parameters: primitives, derived and decided. Figure 2 shows the model of classes used for implementing ADD parametric networks.

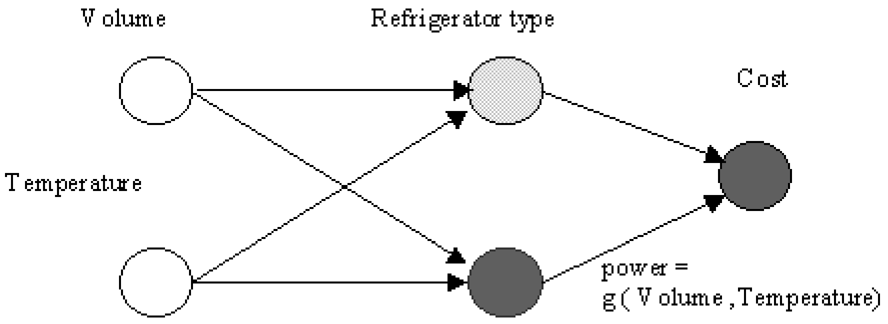

Figure 1 – Parametric Network

Primitive parameters (white circles on figure 1) represent problem requirements. They are used as input parameters and their values must be provided by users. In the parametric network of figure 1, Volume and Temperature are primitive parameters.

The derived parameters (dark circles on figure 1) depend on other parameters. The derived parameter value is calculated by an expression. These expressions may be logic relations or heuristic rules involving other parameters.

For example, the Power parameter is calculated by an expression involving Volume and Temperature parameters. Cost is a function of the Volume and Refrigerator Type parameters.

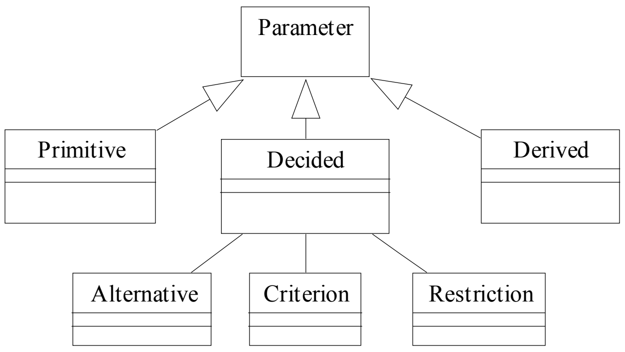

Figure 2 – ADD Model of Classes

The decided parameter (gray circle on figure 1) values are chosen from a set of alternatives by constraint satisfaction and criteria analysis. A decided parameter has a pre-defined set of alternatives. For instance, each alternative of the refrigerator type parameter corresponds to a different kind of refrigerator.

A set of alternatives is filtered by constraints[2]. It means that an alternative is only chosen as a decided parameter value if the conditions established by the restrictions are satisfied by the alternative. For instance, a restriction may specify that the refrigerator volume must be larger than the required volume defined by the user.

Whenever many alternatives satisfy all the restrictions, the decided parameter value must be chosen by comparing the alternatives performance on a set of criteria. A criterion represents the level that the alternatives satisfy a determined condition. For instance, one criterion may specify that the refrigerator should maximize the refrigerated volume. Another criterion may specify that the refrigerator should keep the temperature as close as possible of the required temperature defined by the user.

The interaction between the user and the ADD system is initialized by the user. He proposes primitive parameter values. After proposing values, the user asks the ADD system to compute some derived and decided parameters values. According to the results obtained, the user may change successively the proposed values of the primitive parameters performing an effective search over the problem space.

Sometimes, the user may not understand the reason why the ADD system proposed a specific value to a parameter. At this moment, the user may ask for explanations to the ADD system. The ADD system shows its rationale, i.e., the reasons behind its choice. If the user disagree about the rationale, he may change the ADD system knowledge base for reflecting his way of reasoning. After the end of decision process, the ADD system knowledge base will have enough knowledge to explain the whole decision process rationale.

Figure 3 shows the ADD system architecture proposed by Garcia [1]. The user uses the Design Interface for proposing parameters values and asking suggestions to the ADD system.

An inference machine, called Anticipator, generate the suggestions of solutions. If the user disagree with the solution proposed, he interacts with the Knowledge Elicitor through the Justification Interface for changing de knowledge base in order to eliminate the conflict.

The user may also override the proposed solution directly. In this case, the Reconciler will recognize the disagreement and then will call the Elicitor to resolve the differences between the user and the ADD system.

The user may ask questions to the ADD system using the Explanation interface. In this situation, the Rationale Generator generate the explanations about the decision process accessing the knowledge base and the system log. The explanations are showed to the user using the Explanation Interface.

3 Integrating Search Parameters on ADD Systems

An ADD system is not able to perform automated search in a state space due to expressive limitations of their knowledge representation language. Actually, the only way to do search in ADD systems is through the user proposed values. It means

[2] We call them restrictions.

that the user is responsible for defining the states to be evaluated and the sequence which they are investigated. However, since this is a kind of mixed (manual and automatic) search, the number of alternatives may not be enough for finding a nice solution.

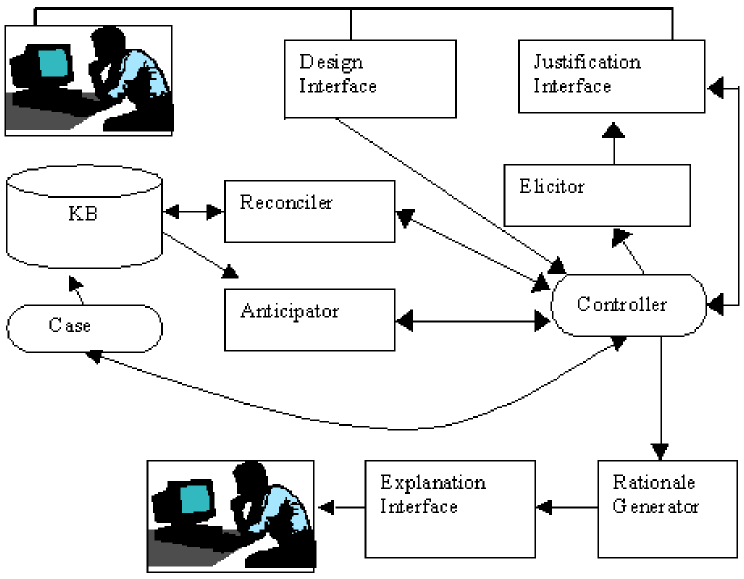

Figure 3 – An ADD System Architecture [1]

For instance, in the example illustrated on figure 1, the user may try to determine the project final cost considering a variation of temperature between 20^{o}C and 26^o C and a variation of space to be refrigerated between 0.5 m^3 and 1.0 m^3. In this example, if the user selects four values to temperature and four values to volume he would test 16 alternatives. If the user wants to test all combinations, he ought to execute the design 16 times, providing the value for each combination (the values of volume and temperature) and asking the system to successively calculate the final cost.

Thus, the generation of combinations to be tested is a user task. In some circumstances this approach is fine because the user may use his heuristic knowledge and common sense for guiding the search.

On the other hand, this option demands a lot of time and effort from the user. This may induce the user to perform only a few assessments, considering the problem space size, and choose the best combination achieved considering the cost criterion. The solution chosen through this technique may be far from the optimal solution once the problem space was not widely investigated.

It would be interesting, on these situations, to allow the definition of a new kind of parameter in the ADD system knowledge base which would enable the system to choose a solution in a problem space through an automated search method. This method would be responsible for generating and assessing the alternatives and choosing the best one considering existing requirements, constraints and preferences. Figure 4 shows a new parametric network for the Refrigerator problem using a search parameter. With this new network, the user only specifies the temperature variation interval and the volume variation interval. The Cost parameter (black circle on figure 4) is a search parameter that allows the ADD system to use those values for generating and assessing the combinations applying a specified method.

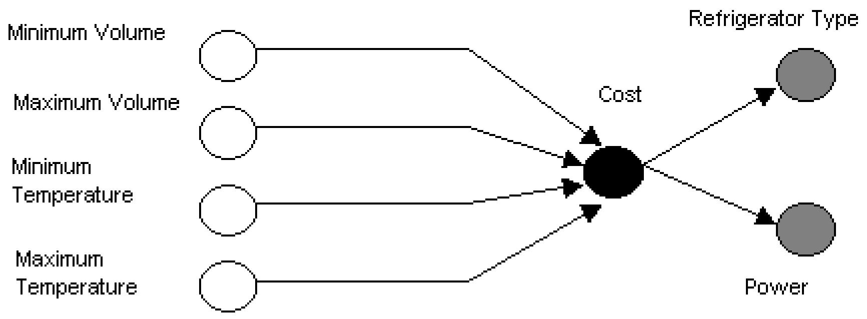

Figure 4 – A Parametric Network with a Search Parameter

It would be even more interesting to let the user dynamically (i.e., during the decision process time) selects the search method to be applied from a method library. In this way, the user may apply his problem specific context knowledge to choose the most adequate method to the particular situation and environment.

3.1 Proposed Model

This work adds a new kind of parameter to the model of classes showed in figure 2, called search parameter. With this new parameter search methods can be applied for solving problems with ADD systems.

The search parameter is based on a method ontology for the parametric design task proposed by Motta [4] to illustrate his TMDA approach ("*Task, Method, Domain, Application*") for construction of reusable knowledge components. In TMDA, a knowledge base is defined by the task, the problem solving method, the domain and the application knowledge.

Our work considers this new kind of parameter as a problem space to be searched by a problem solving method (PSM). Then, for building a search parameter, the knowledge engineer needs to specify the problem space, choose and customize a PSM that will be able to find the problem solution. In order to specify a problem space, the knowledge engineer must supply task knowledge about the problem domain. Task knowledge is provided by the specification of input, functionally bound and key properties and also by the specification of constraints and criteria.

Input properties have values directly obtained from the parameters that the search parameter depends on. Functionally bound properties have theirs values functionally determined by a constraint. Key properties are those which values must be determined by the PSM. These properties define the size of the search space.

Constraints are conditions that must be satisfied by the problem solution. Criteria are functions for comparing problem solutions. Taking into account a range of possible values for each key property, the PSM chooses one value that satisfies all constraints and better satisfies the criteria.

Therefore, besides providing the task and domain knowledge, the knowledge engineer must also provide the PSM to be applied during the search parameter definition.

Instead of building a monolithic PSM library, we followed the approach proposed by Motta and constructed a configurable component library. The knowledge engineer may select various components, configure them and put them to work together instead of having to select a complete method from the library. Figure 5 illustrates the knowledge components used for customizing one PSM.

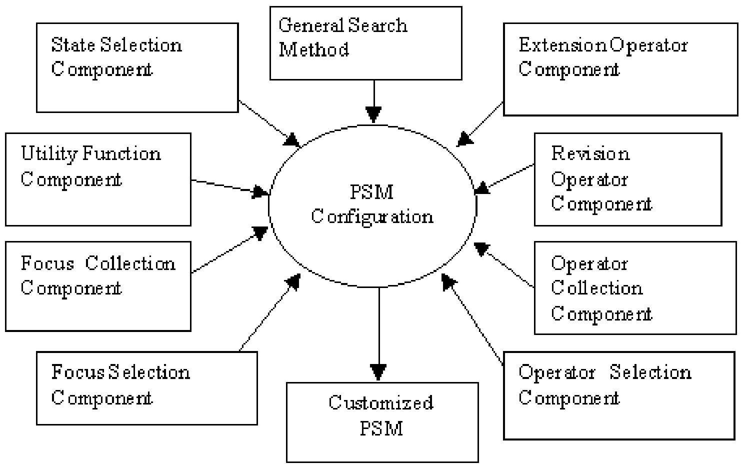
State Selection Component
General Search Method
Extension Operator Component
Utility Function Component
PSM Configuration
Revision Operator Component
Focus Collection Component
Operator Collection Component
Focus Selection Component
Customized PSM
Operator Selection Component

Figure 5 – Configuring a PSM

A General Search Method is a domain-independent abstract method that basically consists of iterative steps for generating alternatives, selecting alternatives and evaluating the selected alternative.

A state is defined by a valued property set. If all properties are valued, the state is called a complete state. If the state is complete and all constraints are satisfied then the state is a problem solution. A State Selection Component implements a state selection strategy. For instance, a state selection strategy could be selecting the most specific state (i.e., the state with the largest number of valued properties).

At a given instant of the problem solving process, the PSM must choose a property to be the process focus. The Focus Collection Component is responsible for choosing a set of properties from which the PSM may select the problem solving focus. For example, the focus collection operation could choose the properties still not valued. The Focus Selection Component is responsible for choosing one property of the property set to be the process focus. For example, the focus selection operation could select the property with the smallest number of possible values.

Once a property has been selected as a focus, the PSM ought to choose one of its values to investigate. Operators are responsible for generating this value. There are two types of operators available: extension and revision operators.

Extension operators are functions associated to properties that suggest the next value to be tested. For instance, one extension operator for the refrigerator type property may return the available equipment with the largest volume in the catalog. Revision operators are linked to constraints. They change property values trying to remove constraint violations. Revision operators are used by methods that perform revisions like the Propose and Revise method. For instance, one revision operator for the refrigerator type property may return the next available equipment in the catalog that is able to refrigerate a larger volume than the volume of the current equipment choice.

The Operator Collection Component selects operators that may be applied to a property on a certain state and context. For example, one component may select all available extension operators which application would certainly not violate the minimum volume constraint. The Operator Selection Component chooses an operator to be applied taking into account a set of state conditions. For instance, one component may select the revision operator that may remove a volume constraint violation.

The Utility Function Component is used for classifying the solutions considering a set of pre-defined criteria. For instance, one component may classify better the value alternatives that minimize the refrigetaror cost and maximize the refrigerator volume.

The integration of a search parameter to a parametric network involves linking it to other parameters. The value of the parameters on which it depends are provided to the input properties whenever the search parameter needs to be calculated. Moreover, the values of the functional-bound and key properties are available to the other parameters of the parametric network after the search parameter is calculated.

After the knowledge base construction is complete, it may be used for supporting a decision-making process. At this moment, it is necessary to get the input properties values and then execute the PSM. If the results obtained by the PSM don’t satisfy the user or if the chosen PSM doesn’t find one solution, then the user may change the PSM choosing other PSM or changing some original method components (for instance, the user may change the state selection strategy or change the focus selection strategy).

4 Conclusions

The representation language extension proposed here allows ADD systems to perform automated search in some specific contexts. In order to test the feasability of our approach, the Hill Climbing (HC), Propose and Backtracking (PB), Complete the Model Then Revise (CMR) and Extend the Model Then Revise (EMR) methods were implemented. The HC and PB methods were used in the Sisyphus I problem solving. Sisyphus I is a room allocation problem proposed by Linster [3]. In this problem, 15 employees are allocated to 10 rooms according to a set of constraints and preferences.

Different HC and PB configurations were made. The difference between configurations was the chosen operator type (generic or problem specific). The configurations were tested in a Pentium II 300 Megahertz with 128 MB de Ram. The time spent always was between 0,3" and 1" except PB2 that spent 11" to get a solution. PB2 generated 10.479 states and performed 7.163 backtracks to find one solution. PB1 generated only 75 states, doesn't performed any backtrack and spent only 0,3" to find a solution. PB2 found the best solution.

The proposed extension allows the utilization of different search methods in ADD systems. These methods are selected from a reusable components library. Moreover, the component-based organization of the library allows the knowledge engineer to use and customize PSMs easier than if he had to create his own methods from scratch. He only needs to select pre-existent components from the library for configuring a PSM.

The user may also choose and dynamically customize the method to be used. Besides, if the customized method doesn't find a solution to the problem or if it doesn't reach a satisfactory solution, the user may try a new configuration, even changing the whole method to be used.

Currently, the user will probably not benefit from the search parameter dynamic customization once it would be necessary to change the code and recompile the ADD system. This is quite obstrusive for the problem solving process. We are developing a generic tool for constructing and using Actice Documents, called JADE (Java-based Active Documents Environment) [5]. One of our next steps is the inclusion of the search parameter to the representation language used by JADE. This will enable the user to change the PSMs during the problem solving session without having to logout, change code and recompile the system.

However, even if JADE incorporates the search parameter, there is one important aspect to be still investigated about the required human user background knowledge. We assume that the user is a domain expert. Once they are not necessarily computer scientists, they probably don't have computational knowledge for choosing the appropriate PSM and its components. For enabling the users to benefit from this type of interaction, the knowledge engineers will have to train the users for applying these features. We also expect that the user will learn from its own experience of using the PSMs to specific problems.

Another point to be further investigated is experimentation. The Sisyphus I experiment was used to exemplify the application of our approach. Since it is not a complex problem, it would be interesting to investigate our approach in more complex domains for verifying the methods efficacy and for checking the independence of the implemented methods from the application domain.

References

1. Garcia, A. C. B., Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design. PhD thesis, Stanford University, United States, 1992.
2. Garcia, A. C. B., Rodrigues, R. F. & Moura, R.: ADDVAC: Applying Active Design Documents for the Capture, Retrieval and Use of Rationale During Offshore Platform VAC Design. American Association for Artificial Intelligence, 1997.
3. Linster, M., Problem Statement for Sisyphus: Models of Problem Solving. International Journal of Human Computer Studies 40(2) pp. 187-192, 1994.
4. Motta, E., Reusable Components for Knowledge Models. PhD Thesis, Knowledge Media Institute, The Open University, UK, 1998.
5. Varejão, F.M., Pessoa, R., JADE: A Computational Environment for Constructing, Using and Reusing Active Design Documents. Proceedings of the 5th IFIP WG 5.2 Workshop on Knowledge Intensive CAD. St. Julians, Malta, p. 99-114, 2002.