An Overview to Knowledge Acquisition

Vikash Verma[1] Upender Yadav[2] Parmod Saha[3] Akash Verma[4]
[1,2,3,4]Computer Science Engineering
[1,2,3,4]Dronacharya College of Engineering

Abstract— In this paper we have described an integrated acquisition platform that includes several techniques previously developed to support users in various ways as they add new knowledge to an intelligent system. As a sum of this integration, the individual techniques can take better advantage of the context in which they are invoked and provide well suggestions to users.

Key words: Integrated Acquisition, Machine learning, Interdependency Models

I. INTRODUCTION

Knowledge acquisition means one can learn by experience and by storing the experience in a knowledge base.one basic example of this type is rote learning. The domain expertise that needs to be transferred to an expert system is a collection of definitions, relations, specialized facts, procedures and assumptions. The transfer of the knowledge from some knowledge source to a computer system is called knowledge acquisition. To acquire knowledge from human experts is known as knowledge engineering.

An important area of user interface research is the development of practical approaches that enable users to add new knowledge to an intelligent system, which would bring computers closer to meeting the challenge of end-user programming. These acquisition interfaces need to have many intelligent capabilities in order to support the complex dialogues that they must conduct with the user, integrate the new knowledge with existing knowledge, and make appropriate generalizations. The paper begins with a brief overview of the individual pieces of knowledge acquisition research that our interface integrates. Next, we analyze the different kinds of knowledge that users need to specify and discuss the challenges they pose to a knowledge acquisition tool. We then describe the components of our implemented system in detail, highlighting the benefits of the integrated acquisition environment.

II. KNOWLEDGE ACQUISTION TECHNIQUES

The **therr models** of knowledge acquisitions defined by Bunchanan and Shortliffe are:

(1) Handcrafting: Handcrafting means Code knowledge is converted into program directly.
(2) Knowledge Engineering: Knowledge engineering means working with an expert system to organize his/her knowledge in a suitable form an expert system to use.
(3) Machine Learning: Machine learning means to extract the knowledge from training examples.

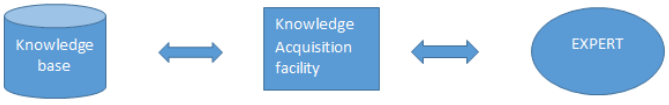

Fig. 1: Knowledge Acquisition Facility

Users do not know formal languages.

- We have developed English-based editors that allow users to modify English paraphrases of the internal, more formal representations [3]. Users can only select the portions of the paraphrase that correspond to a valid expression in the internal language, and pick from a menu of suggested possible replacements for that portion. This approach enables the system to communicate with the user in English while circumventing the challenges of full natural language processing.
- How do users know where to start? Intelligent systems use knowledge to perform tasks for the user. If the acquisition tool has a model of those tasks, then it can reason about what kinds of knowledge it needs to acquire from the user. We have developed acquisition tools that reason about general task models and other kinds of pre-existing knowledge (such as domain-specific knowledge that is initially included in the system's knowledge base) in order to guide users to provide the knowledge that is relevant to those tasks [2]. Our work has concentrated on plan evaluation and assessment tasks, but could be used with other task models.
- How do users know that they are adding the right things? Users need to know that they are providing knowledge that is useful to the system and whether they have given the system enough knowledge to do something on its own. Our approach is to use Interdependency Models that capture how the individual pieces of knowledge provided work together to reason about the task [10]. These Interdependency Models are derived automatically by the system, and are used to detect inconsistencies and missing knowledge that turn into follow up questions to the user. Users are often not sure whether they are on the right track even if they have been making progress, and we have found that it is very useful to show the user some aspects of this Interdependency Model (for example, showing the sub steps involved in doing a task) and how it changes over time.
- How do users figure out what to do next? A system that responds to the user with many sensible follow-up questions is helpful but not sufficient. When users add a sizeable amount of knowledge, the system is likely to respond with a number of follow up questions. Users also need help in formulating answers. We have categorized and organized different kinds of questions, as well as the possible actions that the user can take in order to address them [10, 8]. By understanding the nature of the questions, the system can help the user understand and prioritize these questions. By

understanding the context in which each question was generated, the system can make good guesses about how the user may answer them.

- It takes several steps to add new knowledge, so users will be easily lost. Entering new knowledge, even of moderate complexity, requires making several individual changes. Users often do not realize and/or forget the side effects of these individual changes that must be followed up (anyone who has ever programmed can relate to this). We have developed a framework to guide users through typical sequences of changes or Knowledge Acquisition Scripts (KA Scripts) [18]. As the user interacts with a KA Script and enters knowledge step by step, additional KA Scripts may be activated that contain follow-up questions about that new knowledge. The system not only points out the next steps but indicates its best guess about the knowledge the user needs to enter based on the knowledge that is already in the system. We have developed a library of general-purpose KA Scripts [18], as well as with scripts customized to the general task models mentioned above [2].

III. What Kinds of Knowledge Need to Be Acquired?

In terms of the knowledge acquisition techniques that we believe are needed, we distinguish three main categories of knowledge:

- Data denotes specific object instances or constants (e.g., UA flight 22 departs from LAX at 12:00PM and arrives Madrid at 9:20AM.). This is perhaps the easiest for users to specify. They can be effectively acquired through form-filling interfaces. In the extreme, some of this data acquisition can become extremely complex, since in some of the applications we have seen object instances that are composed of several hundred assertions. Model-based interfaces that declaratively represent object classes and the associated interfaces have been used to acquire object instances of medium complexity [15]. The design of effective tools to acquire complex object instances remains largely an open research issue, in our view one that has more to do with the HCI aspects of the interface rather than providing more intelligent assistance.
- Object classes refer to general descriptions of object categories and the relations that exist among them (e.g., a flight has an origin and a destination). This is sometimes called ontology. A variety of ontology editors have been developed over the years [1, 19, 6, 16, 9], many of them include facilities to enter data as well. Other tools have focused on related issues such the elicitation of attributes and differentiating features [7, 4], and the detection of inconsistencies in ontological descriptions [13]. Although many of these tools use graphical interfaces to show class hierarchies and relations, it appears that hypertext interfaces may be more practical for sizeable knowledge bases.
- Constraints and preferences specify how the user would like the system to make choices in cases when there are alternative options. This is perhaps the most difficult kind of knowledge to acquire, and has been the main focus of our work. Consider the example of renting a car only when using taxis is more expensive. This requires finding the distances between airport, hotel, and meeting locations to estimate the cost of taxis, figuring out how many days the car is rented to estimate the cost of the rental car, and comparing the two total amounts. This is problem solving (procedural) knowledge, which in this case is reasoning with the information provided (e.g., the meeting location) in order to derive more complex abstractions (e.g., the total cost of taking taxis). Problem solving knowledge is quite difficult to acquire, and is the end-user programming challenge as described in [5]. Because it uses information from the object classes and the data, it needs to be specified in a way that is consistent with those. Because it is generating intermediate abstractions and using them to generate more complex ones, it involves a number of steps and sub steps that have to fit together precisely. We aimed to address these issues in our past work, especially the research on Interdependency Models [10, 8]. Alternative approaches learn from specific examples provided by users as they perform a task, and include programming by demonstration, learning apprentices, case-based reasoning, and feature-based induction.

IV. Integrating Knowledge Acquisition Techniques

Figure shows an overview of the different interface components that are integrated within our system and how they interact with each other. Many of the component tools become more powerful and easier to use in the integrated system because of the information that is shared between them

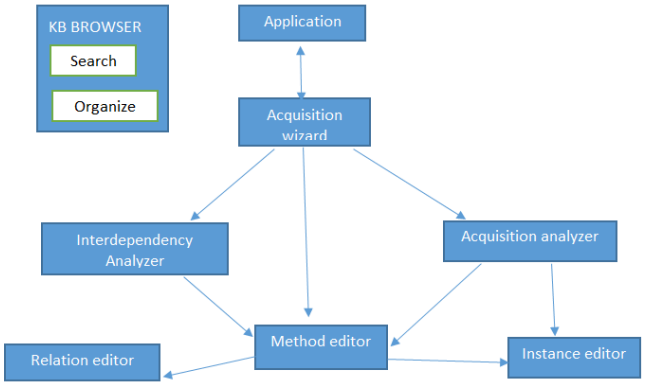

Fig. 2: Overview of the Integrated Interface Components and Their Interactions

V. Acquisition Wizard

When the acquisition tool is invoked from the application, the Acquisition Wizard manages the initial interaction with the user shown in Figures 2 and 4. The wizard uses a general task model of plan evaluation that represents general classes of constraints and properties, and uses KA Scripts to organize the questions that it needs to ask the user in order to classify the new constraint appropriately. This task model includes an ontology that organizes different classes of plan

evaluation criteria, shown in Figure 10. The task model also includes problem solving knowledge about how each class of criteria is evaluated. Although the ontology shown in Figure 10 is not complete, and there will be cases where it does not help the user, it provides valuable guidance in many cases. A more inclusive ontology that includes planning resources can be found in [2], as well as more details about the techniques used. The ontology shown here divides the set of evaluations for which the wizard can provide help in two different ways. First, the evaluation can be either global, meaning that the property is computed once for the whole plan (e.g., the total cost) or it can be local, meaning that the property is computed for each one of a set of objects found in the plan (e.g., the length of each flight).

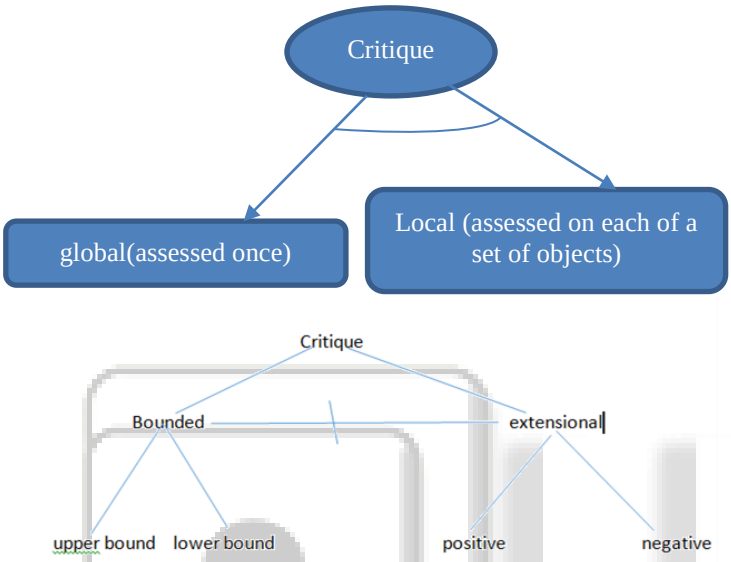

Fig. 3: Part Of The Ontology Of Types Of Plan Evaluation Used By The Acquisition Wizard To Classify And Provide Help For A New Critique.

VI. Method Editor for Problem Solving Knowledge

The Method Editor is invoked to help the user add problem solving knowledge. It can be invoked by the Acquisition Wizard (as is done in our example for the windows in Figures 3, 5, and 6) which uses the general problem solving knowledge in the task model to create an initial template of the method. The method editor can be invoked in other contexts (from the Interdependency Analyzer and from the Acquisition Analyzer as we will describe below), and in all cases the editor uses that context to generate an initial template of the method. It can also be invoked by the user directly to update a method created previously. The method editor is always invoked with some initial template or specification of the problem solving knowledge to be added by the user. At the top of the window, the method editor displays an automatically generated English description of that initial specification of the problem solving knowledge. The editor keeps track of what sub expressions in the formal language result in each substring of the English paraphrase (for example, (r-hotel?) results in the hotel of the step). The paraphrase is mouse sensitive, but the user can only select meaningful portions of it. When the user has selected the part of the paraphrase that she wants to change, the method editor analyzes the formal sub expression that generated it and displays a tree view of the possible alternatives, in English, in the lower window. The alternatives are automatically generated by an algorithm that ensures that choosing any one would maintain the syntactic consistency of the method, as shown in [3].

VII. Relation (and Concept) Editor

When the alternatives presented by the method editor do not include what the user would like to specify, the user can select "other" and a Relation Editor is invoked. In our example, this was done in Figure 6 when the user found that contracts did not have maximum allowed hotel rates. Our integrated tool allows the user to switch seamlessly between adding problem-solving knowledge and extending the descriptions of objects in the knowledge base, which is often necessary. An additional benefit of the integrated tool is that it can use the context (in this case the method being edited) to propose a default domain and range for the new relation (contract and number), further simplifying the user's task. Our integrated interface does not yet include a concept editor, although it would be integrated in a similar way.

VIII. Acquisition Analyzer

Throughout the interaction with the user, an Acquisition Analyzer keeps track of all the pending questions. Many of these questions result from analyzing the Interdependency Model derived by the system, as described in [10, 8]. The Acquisition Analyzer keeps track of the reason for each question to the user. For example, the Interdependency Model states what information about objects is needed to check the new constraint. In our example, it will notice that the maximum allowed hotel rates of contracts is used in the constraint's definition, whereas the end dates or funding agencies of contracts are not. Based on this, the Acquisition Analyzer will create a question for the user about each existing contract.

IX. Instance Editor

The Instance Editor is invoked when the user needs to enter information about particular objects, such as a new location or meeting. In our example, the Instance Editor is used to specify the maximum allowed hotel rate of existing contracts, as shown in Figure 7, and could be invoked from the Acquisition Analyzer's agenda. The Instance Editor shows on the left hand side what information is needed about the particular object to check the constraints and properties defined by the user so far. On the right hand side it gives the user the option to specify additional things, but it notes that these are not currently needed. The editor makes this distinction by analyzing the Interdependency Model which is maintained by the Interdependency Analyzer as the system acquires new problem solving knowledge. This feature, enabled by our integrated system, is not provided by other instance editors.

X. Interdependency Analyzer

So far, the components of our acquisition interface that have been described are fairly easy to use, but the information they acquire is relatively simple. More advanced users may explore an additional component of the interface that enables them to enter more complex constraints. To support the acquisition of more complex constraints, users need to

add substantial amounts of problem solving knowledge as we discussed earlier. The Interdependency Analyzer guides the user in entering problem solving knowledge for more complex constraints.

XI. CONCLUSIONS

We have described an acquisition interface that integrates previously developed techniques to guide users in different aspects of knowledge acquisition. Compared with the individual techniques, it is able to make better use of the context of the user's actions, so the resulting interface provides stronger guidance to users.

REFERENCES

[1] G. Abbrett and M. Burstein. The kreme knowledge editing environment. International Journal of Man-Machine Studies, 27(2):103–126, August 1987.
[2] J. Blythe. Extending the role-limiting approach: Supporting end users to acquire problem-solving knowledge. In ECAI 2000 Workshop on Applications of Ontologies and Problem-Solving Methods, 2000.
[3] J. Blythe and S. Ramachandran. Knowledge acquisition using and english-based method editor. In Proc. Twelfth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, 1999.
[4] J. H. Boose and J. M. Bradshaw. Expertise transfer and complex problems: Using aquinas as a knowledge acquisitiong workbench for knowledge-based systems. International Journal of Man-Machine Studies, 26, 1987.
[5] A. Cypher. Bringing programming to end users. In Watch What I Do: Programming by Demonstration. MIT Press, 1993.
[6] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: A tool for collaborative ontology construction. In Proc. Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, 1996.
[7] B. R. Gaines and M. L. G. Shaw. Eliciting knowledge and transferring it effectively to a knowledge-based system. IEEE Transactions on Knowledge and Data Engineering, 5(1), 1993.
[8] Y. Gil and E. Melz. Explicit representations of problem-solving strategies to support knowledge acquisition. In Proc. Thirteenth National Conference on Artificial Intelligence. AAAI Press, 1996.
[9] P. D. Karp, V. K. Chaudhri, and S. M. Paley. A collaborative environment for authoring large knowledge bases. Journal of Intelligent Information Systems, 13:155–194, 1999.
[10] J. Kim and Y. Gil. Deriving expectations to guide knowledge-base creation. In Proc. Sixteenth National Conference on Artificial Intelligence, pages 235–241. AAAI Press, 1999.
[11] J. Kim and Y. Gil. Acquiring problem-solving knowledge from end users: Putting interdependency models to the test. In Proc. Seventeenth National Conference on Artificial Intelligence. AAAI Press, 2000.
[12] G. Linden, S. Hanks, and N. Lesh. Interactive assessment of user preference models: The automated travel assistant. Sixth International Conference on User Modelling, 1997.
[13] D. L. McGuinness, R. Fikes, J. Rice, , and S. Wilder. The chimaera ontology environment. In Proc. Seventeenth National Conference on Artificial Intelligence. AAAI Press, 2000.
[14] K. Myers. Strategic advice for hierarchical planners. In Proceedings of the International Conference on Knowledge Representation, 1996.
[15] A. R. Puerta, J. W. Egar, S. Tu, and M. A. Musen. A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools. Knowledge Acquisition, 4(2):171–196, 1992.
[16] B. Swartout, R. Patil, K. Knight, and T. Russ. Towards distributed use of large-scale ontologies. In Proc. Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, 1996.
[17] W. R. Swartout and Y. Gil. Expect: Explicit representations for flexible acquisition. In Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, 1995.
[18] M. Tallis and Y. Gil. Designing scripts to guide users in modifying knowledge-based systems. In Proc. Sixteenth National Conference on Artificial Intelligence. AAAI Press, 1999.
[19] L. G. Terveen and D. A. Wroblewski. a collaborative interface for browsing and editing large knowledge bases. In Proc. Eighth National Conference on Artificial Intelligence. AAAI Press, 1990.

