reprinted from
IEEE EXPERT
Intelligent Systems and their Applications

Interview-Based Knowledge Acquisition Using Dynamic Analysis

Atsuo Kawaguchi and Hiroshi Motoda, Hitachi
Riichiro Mizoguchi, Osaka University

USING DOMAIN- AND TASK-INDEPENDENT PRIMITIVES CAN FACILITATE OUR WRITING TASK-SPECIFIC INTERVIEW STRATEGIES. USING BOTH STATIC AND DYNAMIC ANALYSIS LETS US BEGIN KNOWLEDGE ACQUISITION WITH AN INCOMPLETE DOMAIN MODEL AND LATER REFINE AND BUILD THE KNOWLEDGE BASE. WE CAN ACQUIRE KNOWLEDGE FOR CASES WE HADN'T EVEN THOUGHT OF AT THE TIME OF STATIC ANALYSIS.

INTERVIEWING IS ESSENTIAL IN eliciting new knowledge from domain experts. However, computer-based knowledge-acquisition systems suffer from the so-called knowledge-acquisition dilemma: If the system is ignorant, it cannot raise good questions; if it is knowledgeable enough, it does not have to raise questions. Most efforts have therefore been devoted to identifying what knowledge to give a system in advance and how to use that advance knowledge to facilitate acquisition.[1]

The tools for building expert systems are no longer simple, unstructured, general-purpose ones. Today's tools are designed for specific tasks, such as diagnosis or design.[2] Accordingly, many interview systems for knowledge acquisition are designed to elicit knowledge for a particular class of problems. They use a task structure (that is, a problem-solving method) to guide knowledge acquisition.[3-5]

Interviewing itself forms a class of tasks, and it is worthwhile to look into the kind of knowledge that a good interviewer uses. Interviewers have knowledge about the task and domain under consideration, and about interviewing itself. It is the latter knowledge that makes the interviewer an interviewing expert. Since interviewing (like diagnosis or design) is a class of tasks, we can represent its task structure with a set of primitives that are independent of both domain and application tasks and are only interview-task dependent. These primitives can facilitate the building of interview systems. Existing knowledge-acquisition tools, however, emphasize embedding knowledge of the problem-solving methods.

Our major objective was to identify a set of interviewing primitives. We have implemented this idea in a prototype system called the Shell for Interview Systems (SIS),[6] from which we can generate application-specific interviewing systems. Using the primitives and other information on application tasks and domains, we can write task-specific interview strategies. Our prototype is therefore an interview metasystem (a system that generates an interview system).

This is certainly not a new idea. For example, Protégé is a tool for creating models.[5] Assuming a class of tasks that can be solved using skeletal-plan refinement, Protégé generates application-tailored, model-extending tools such as Opal, a knowledge-acquisition tool for the Oncocin medical expert system.

In contrast, our approach does not embed any problem-solving method in advance. In other words, Protégé uses a top-down approach whereas our prototype takes

Table 1. Primitive attentions for static analysis.

Primitive attention	Activation criterion
NosameFrameNosameSlotNosameValue	Nothing to share
SameValue	Same value to share
SameSlot	Same slot to share
SameSlotSameValue	Same slot and its value(s) to share
SameFrame	Same frame (name) to share
SameFrameSameValue	Same frame and value to share
SameFrameSameSlot	Same frame and slot to share
SameFrameSameSlotSameValue	Same frame, slot, and value to share

a bottom-up approach. Having a task model is certainly advantageous, because no interview is feasible without knowing how the knowledge is to be used. On the other hand, since similar tasks use similar strategies, constructing strategies from fundamental primitives builds accumulating libraries, with the same effect as having various task models in advance. Both approaches aim at the same goal but from opposite directions. Although Protégé clearly distinguishes between building and extending models, our examples illustrate that these two activities are relative. One mechanism can generate both tools.

We have demonstrated this idea's generality and effectiveness for static analysis in two implementations. First, we used SIS to generate SIS-More, a program that functions like More, a knowledge-acquisition system for diagnostic expert systems.[3] We also built IIS-LD, an intelligent interview system for the logical design of databases.[7] However, we wanted to extend our ideas to a more general framework by including dynamic analysis, since a system can acquire knowledge more efficiently by actually solving a problem using acquired knowledge and then refining it during problem solving. Therefore, just as More was extended to Mole (the knowledge-acquisition system for diagnostic tasks that first made explicit the idea of knowledge acquisition by dynamic analysis[8]), so we extended IIS-LD to become IIS-DB, an intelligent interview system for database construction that uses a set of additional primitives to describe its interviewing strategy in dynamic analysis.

SIS, an interview metasystem

SIS serves as an interview system skeleton. It provides interview-task-specific primitives and a set of descriptors to represent task-specific interview knowledge. By instantiating SIS with this knowledge, we can obtain a task-specific interview system.

SIS supports knowledge acquisition by static analysis. It specifies question strategies by analyzing the nature of the task in advance, separate from actual problem solving. In other words, the system considers only static attributes of the knowledge base rather than behaviors that result when the knowledge base is actually used. Task-specific interview systems generated by SIS do not contain a problem solver, that is, a component that evaluates the performance of the expert system for which the knowledge is to be elicited. (The version of SIS we describe here is newer than reported previously[6] and uses different terminologies to describe primitives.)

An interview system generated by SIS begins by requesting initial information to start an interview and constructs an incomplete domain model in network form. It then parses an expert's sentences and generates a set of task-specific *attentions*. Based on these attentions, the system raises questions to elicit new knowledge. The domain model is refined through answer/attention/question cycles.

The domain model. SIS represents an application task's domain model as a network with nodes and links, that is, a directed graph. Nodes represent concepts and link relations among concepts. Both nodes and links are represented as frames with slots, and each slot has a set of values. This representation includes standard features such as class hierarchy and class-instance relations.

Generic attentions. SIS's interview-task-specific primitives, called generic attentions, are clues encountered during the course of interviewing that guide the interview flow, prevent unnecessary queries, and make elicitation of new knowledge more efficient. They are similar to the concepts of impropriety, critic, or credit assignment.[9] "Generic" means that these attentions are domain independent and task independent. They are abstract entities that must be instantiated to a specific situation and interpreted as meaningful task-specific attentions when SIS is applied to a specific task.

The purpose of static analysis is to create initial domain models, which are often incomplete. Therefore, we emphasize building models from scratch. (This is not strictly correct, since basic concepts must be provided as prototypes, as we explain later.) The main operation here is to structure the network by comparing each frame with other frames in the partial structure or with known prototype frames. Since both nodes and links are represented as frames, and each frame can be defined by specifying its slots and their values, we have eight $({}_3C_0 + {}_3C_1 + {}_3C_2 + {}_3C_3)$ basic primitives for comparison. We call these *primitive attentions*, and they are listed in Table 1.

Comparing frame structures involves one or more primitive attentions. We define each comparison as a *generic attention* because the result of this operation can be a clue to take a new action. For example, checking whether there is a linkable path between two frames involves looking for a node-frame with a slot whose value equals that of a given slot of a given frame. The generic attention of this operation is FoundCandidate. This is equivalent to comparing two frames in which only one slot value is constrained and shared, which we denote as the primitive attention SameValue. Checking if multiple paths exist from one node to other nodes generates the generic attention called SameSource. This is equivalent to comparing two link-frames that have the same node value for the from-slot, which we denote as the primitive attention SameSlotSameValue.

A sequence of linkable paths between two frames forms a path, and if two paths share one part we say that there is a common path. Checking if there is a common path for different nodes generates the generic attention CommonPath, which can be realized by using the above two primitive attentions in combination. Also, one primitive attention can correspond to more than one generic attention. For example,

Table 2. Generic attentions (thus far identified).

Generic attention	Primitive(s)	Activation criterion
FoundCandidate	SameValue	A concept linkable to a path
SameSource	SameSlotSameValue	Multiple paths to a node
CommonPath	SameValue, SameSlotSameValue	A common path for different nodes
ScriptTracer	SameFrameSameSlot	A query description in the class
UnknownObjectDetector	NosameFrameNosameSlotNosameValue	Unknown concept provided
DiscordDetector	SameFrameSameSlot	Discrepancy found in the slot values of two nodes
AmbiguityDetector	SameFrameSameSlot	More than two values assigned to one slot
CertaintyManager	SameFrameSameSlotSameValue	Reliability within the tolerance
ConstraintChecker	SameFrameSameSlot	Constraint violated for the slot value of a node

checking for a discrepancy between the slot values of two nodes generates Discord Detector; checking if the slot value of a node violates its constraint generates Constraint-Checker. Both are generated by the Same-FrameSameSlot primitive attention. We distinguish between the two generic attentions by specifying which frames or slots to focus on. The "same" here means that the subsumption relation holds. For example, comparing an instance with its class is part of the SameFrame primitive attention.

Table 2 lists the generic attentions we have identified for static analysis. We do not claim that the list is complete. It is only a subset of the generic attentions derivable from primitive attentions. Each generic attention has a corresponding criterion for it to be activated. The primitive attention SameFrameSameSlot is used most frequently because it is a fundamental operation to check a new frame.

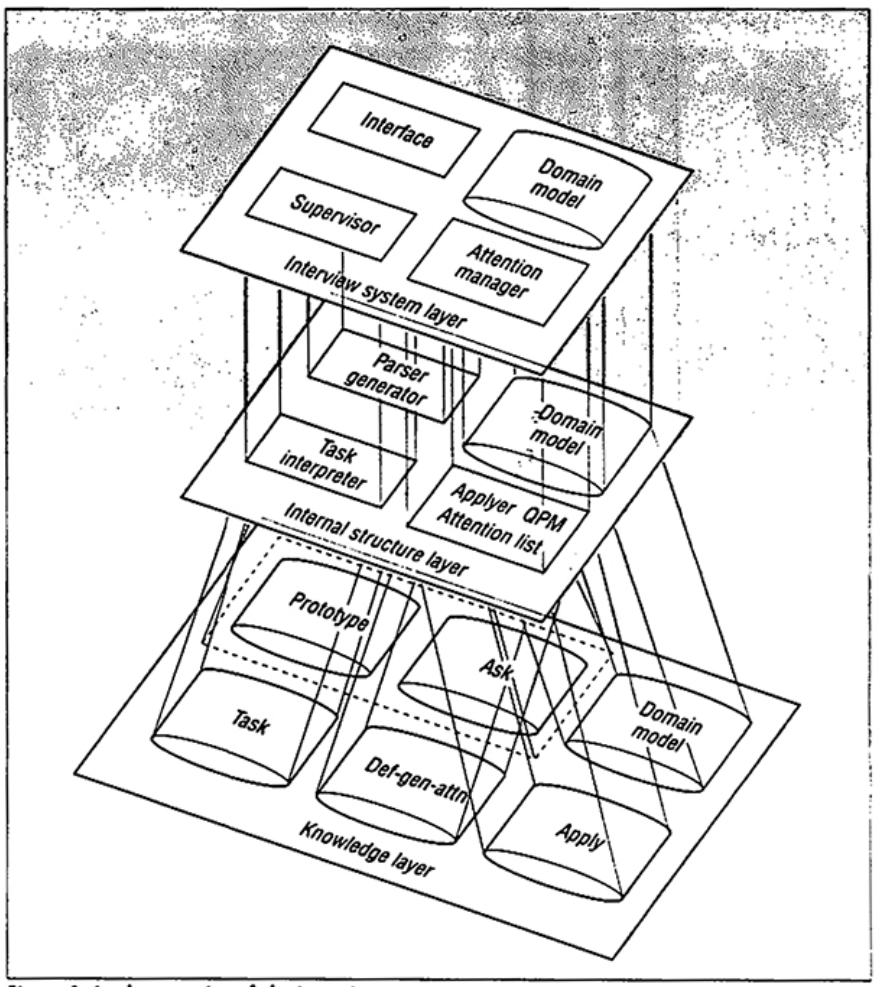


Figure 1. Implementation of the interview system.

Implementation. The top layer of an implemented system is the instantiated interview-system layer (see Figure 1). Below are two more levels. The middle level, the internal-structure layer, represents the system in terms of the program modules specific to SIS. At the bottom, the knowledge layer specifies the behavior of each module in the internal-structure layer by declaratively describing those behaviors. The knowledge given here eventually determines the behavior of the interview system.

Interview strategy. As is clear from the above description, generating and processing attentions are central to knowledge acquisition by interviewing in SIS. The system executes each question and answer by generating a corresponding attention and processing it accordingly. Each attention has its own knowledge to raise a query to the user. The details of the behaviors that depend on each interview system are defined by the knowledge given to each module at the knowledge layer. We use the following descriptors to express five kinds of knowledge:

(1) *Prototype* defines a metadescription of the domain (for example, a class description). It defines the attributes that the nodes and links can have, their default values, is-a relations between nodes, how the knowledge acquired by the interviewee is stored in the domain model, and how the internal structure (domain model) is translated into English sentences.

```
Prototype hypothesis
   super_class noun
   slots
      [link_to_symptom, frequency_cond].

Prototype symptom
   super_class noun
   slots
      [link_to_symptom, link_to_hypothesis, link_to_test].

Prototype s_h_link
   super_class noun
   slots
      [symptom, hypothesis, condition, attribute, value].
```

Figure 2. SIS-More Prototype definitions.

```
Task construct_KB_in_More_manner
   sub_task
      make_initial_domain_model,
      while attentionListNotNil
      do check_domain_model,
         generate_rule_set,
         check_rule_set.
Task make_initial_domain_model
   sub_task
      ask_initial_set_of_symptoms,
      ask_initial_set_of_hypothesis,
      ask_initial_association.
Task check_domain_model
   sub_task
      while getAttention → A
      do process_attention (A).
```

Figure 3. The top-level task of SIS-More.

```
Def-gen-attn SameSource
   use SameSlotSameValue
   object_pattern F1 → s & F2 → s   /* Pattern used in Apply */
   attention_pattern
      SameSource(F1.list_of(F2)).
```

Figure 4. Def-gen-attn.

```
Apply
   SameSource
      object
         domain (s_h_link) → symptom & /*Find two s_h_links that have */
         domain (s_h_link) → symptom.  /* the same symptom value.    */
```

Figure 5. Apply.

```
Task differentiation
   attention SameSource (L1, L2_s)
      sub_task
         Symptom = L1 → symptom,
         callProlog getPathEndList (L2_s, EndList),
         callAsk ask_differentiation_start (Symptom, EndList),
         differentiation_ask( Symptom, EndList, HypoNot),
         when HypoNot = [H1,H2|_]
         do symptomDistinction (Symptom, HypoNot).

Task differentiation_ask (Symptom, EndList, ResultList)
      sub_task
         allocVariable(RR),
      (  while getOneElement(EndList) → H
         do callProlog remove(H,EndList,HR),
            callAsk ask_new_symptom_by_diff (Symptom,H,HR,R),
            when R !== [] do pushToVariable(R,RR)
      ). getVariable(RR) → ResultList,
         freeVariable(RR).
```

Figure 6. A Task description of differentiation.

(2) *Def-gen-attn* defines generic attentions from primitive attentions. The user can add generic attentions to the list that SIS recognizes.

(3) *Apply* defines where to apply a generic attention. It specifies which objects to focus on and detects a network location where the system adds, modifies, or deletes already acquired knowledge. The generic attention can be instantiated to a task-specific attention through renaming. The system generates this attention only when the corresponding activation criterion is satisfied for the objects defined by the Apply declaration. Each time the domain model changes, the system searches automatically to see if any criterion is satisfied for each object specified by this declaration.

(4) *Task* controls the interview flow. It also defines how to process generic or task-specific attentions when they are activated. It can handle loops, conditional branches, and subtask calls. It also calls Ask, which predefines the question to ask. In other words, the user can define an interviewing strategy with a pair of Apply and Task declarations. Each Task has its own name, which is a strategy.

(5) *Ask* describes queries.

Applications. The interview systems we generated with SIS — SIS-More and IIS-LD — are different in two ways. First, SIS-More's task is diagnosis, while IIS-LD's is design. Second, the systems have different levels of knowledge acquisition: using Musen's terminology,[5] SIS-More extends the task model while IIS-LD creates it. The task model in SIS-More is given as a set of interviewing strategies and a domain model structure (a domain metamodel defined by Prototype descriptors), both of which are the input to SIS. (A domain metamodel is a model that knows about and generates a specific domain model.) SIS-More then acquires instances of diagnostic knowledge for a specific application (for example, the domain of drilling fluids). In IIS-LD, the task model is not the input; it must be acquired. Its input is a kind of metaknowledge to acquire such a model. IIS-LD does not acquire instance data (the content of the designed database).

SIS-More

The task of SIS-More is heuristic classification. The system acquires diagnostic knowledge as rules that relate symptoms to a possible hypothesis. The eight question strategies in More are based on insights made in advance, so that actual problem solving is unnecessary (static analysis). For example, the differentiation strategy is used to distinguish two hypotheses for which the same symptom appears. Unless this strategy is used, cases might arise in which two hypotheses are derived for a single symptom. This kind of problem can easily be predicted in advance and does not require problem solving.

The domain model. SIS-More's domain model is a network of causal relations between symptoms and hypotheses. Since frames represent both nodes and links in SIS, each hypothesis, symptom, and causal relation is represented by an instance of a corresponding class frame, which is defined using the Prototype descriptor.

Interview strategy. We can represent all eight question strategies as well as task control in SIS-More using the five SIS descriptors. Figure 2 shows how we use Prototype to define hypothesis, symptom, and causal link. SIS-More's top-level task, Construct_KB_in_More_manner, updates and checks the domain model until there are no more attentions (see Figure 3). In Figure 4, the Def-gen-attn descriptor defines the generic attention SameSource using the primitive attention SameSlot-SameValue. SameSource, which can be used to represent the differentiation strategy in the task description, is activated if there are multiple paths between a symptom and several hypotheses. Since SameSource stands for the situation of this attention, no renaming to a task-specific attention is necessary. The Apply descriptor in Figure 5 defines the objects on which to focus the SameSource attention. Figure 6 is a Task description of the differentiation strategy. If this attention is generated, the system queries whether there are any symptoms that can differentiate the hypotheses. If multiple hypotheses remain that are not distinguishable, the system queries whether any symptom attributes can help differentiate these hypotheses.

```
I?-interview.
Please tell me any symptoms you know in this domain.
>increase_in_viscosity.
Please tell me all the hypotheses in this domain.
>shale_contamination, water_influx.
Please select symptoms associated with shale_contamination from:
   1. increase_in_viscosity.
>1.
Please select symptoms associated with water_influx from:
1. increase_in_viscosity.
>1.
shale_contamination and water_influx can give the same symptom
increase_in_viscosity. Can you provide a symptom associated with
shale_contamination that cannot be explained by water_influx ?
>no.
Can you provide a symptom associated with water_influx
that cannot be explained by shale_contamination ?
>yes.
Enter the symptoms associated with water_influx.
>increase_in_unemulsified_water.
May increase_in_unemulsified_water be a hypothesis to
increase_in_viscosity ?
>no.
Ok. I will make a rule KB. Wait a moment, please.

If increase_in_viscosity, Then shale_contamination
If increase_in_viscosity, Then water_influx
If increase_in_unemulsified_water, Then water_influx
```

Figure 7. A dialogue with SIS-More. User input is in bold type.

Dialogue. Figure 7 is an example dialogue between a user and SIS-More. The system notices the ambiguity in the relations between the initial symptom and the hypotheses, and raises questions to differentiate the hypotheses. It then asks if the newly provided symptom can be a cause of the already known symptom, and it produces three diagnostic rules.

IIS-LD

IIS-LD creates logical designs of databases based on the entity-relationship model. The task therefore is to enumerate the necessary and sufficient entities in the domain of interest and the relationships (activities) among them. (Entities and relationships here are abstract concepts. Examples are "part-1" and "construct-1," which still need to be instantiated by actual data.)

The domain model. We again use a network to represent IIS-LD's domain model as a plan structure (explained below). Each relationship and entity in the domain is represented by a verb frame and a noun frame. The meaning of the nouns is domain specific, so that each noun is treated as an unknown concept. IIS-LD is not provided with a dictionary for the nouns, but with 17 abstract concepts such as AbstractRelation, PhysicalObject, LivingThing, and so on. Each new noun acquired by the interview is placed at a lower level in the noun hierarchy as a more specific concept. Therefore, each noun frame has its superconcept as well as its attributes (slots and values).

In contrast, IIS-LD has a domain-independent verb dictionary. Each verb frame has slots where the candidates of the noun concept that fit into these slots are

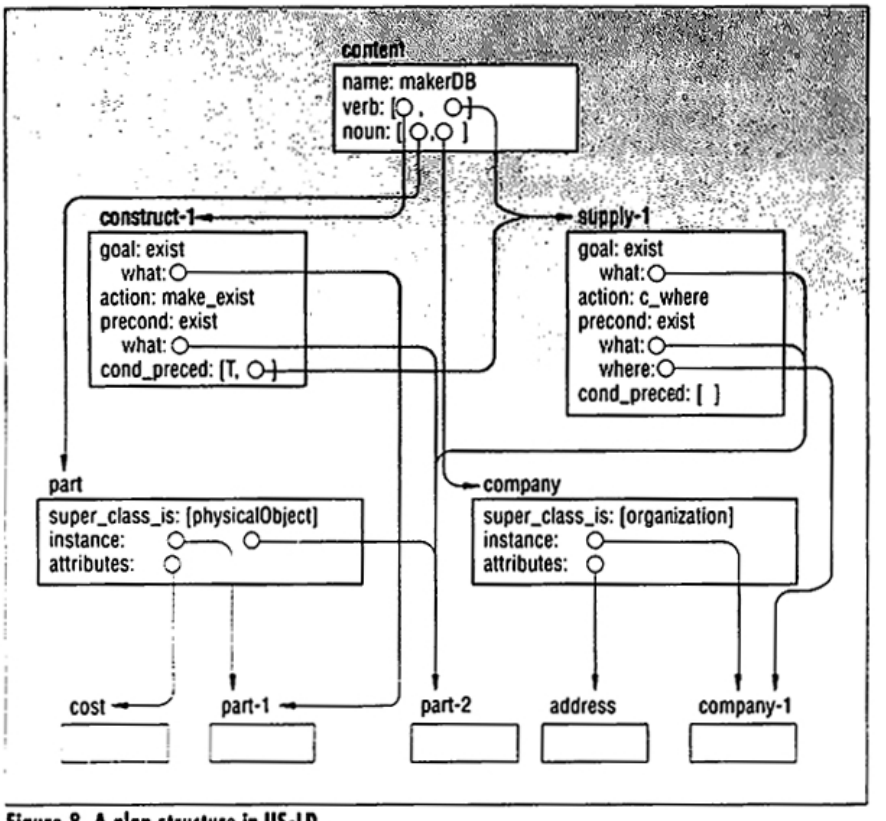


Figure 8. A plan structure in IIS-LD.

```
Prototype construct
   super_class verb
   slots [
      [goal, [exist [what, how_many, when, where]],
      [action, [make_exist]],
      [precondition, [exist, [what, how_many, when, where]]]].
```

Figure 9. An IIS-LD Prototype definition.

filled. Their initial values are abstract concepts. What we give SIS in the first place depends on the nature of each application. In this case, a relationship can be expressed using a verb, and a verb has its own meaning. Thus, it makes sense to prepare the verb dictionary in IIS-LD as part of the domain metamodel using Prototype descriptors.

Plan structure. A sequence of activities forms a plan. If the goal of verb *A* is a subset of the precondition slot (see below) of verb *B*, then we say that *A* and *B* can be linked. A set of verbs linked sequentially is called a plan. Planning constitutes searching for the verb that succeeds or precedes the given plan. Thus the plan structure represents a domain model in IIS-LD. (Correspondingly, the causal structure represents the domain model in SIS-More.)

Figure 8 shows a plan structure, the conceptual diagram of the domain model acquired and used by IIS-LD. Rectangles represent frames. From the content frame, which lists the verbs and nouns used in the domain, we can see that the name of the database is "MakerDB" and that there are two verbs and two nouns. Two frames represent the activity in which a part (part-1) is constructed from other parts (part-2) that are supplied by a company. Each verb

```
Task design_db
   sub_task
      make_initial_domain_model
      generate_ER_model.

Task make_initial_domain_model
   sub_task
      do ask_query -> Q
         check_domain_model
      while Q !== [].

Task check_domain_model
   sub_task
      while getAttention → A
      do process_attention (A).
```

Figure 10. The top-level task of IIS-LD.

```
Task planning
   attention FoundCandidate(F1, F2_s)
   sub_task
      callAsk exist_link(F1, F2_s, Cand),
      callAsk new_value(Cand).

Ask new_value([V1|V_rest])
   showNI ('Please enter an information requirement about ', V1),
   get_answer Query,
   new_value (V_rest).
```

Figure 11. Planning.

```
Def_gen_attn FoundCandidate
   use SameValue
   object_pattern F1 → S1 & F2 → S2
   attention_pattern
      FoundCandidate (F1, list_of (F2)).
```

Figure 12. Def-gen-attn in IIS-LD.

```
Apply
   FoundCandidate
      object
         domain (verb) → precondition &   /* precondition of an instantiated */
         prototype (verb) → goal,         /* verb and goal of the verbs */
                                          /* in Prototype */
         domain (verb) → precondition &   /* precondition of an instantiated */
         domain (verb) → goal             /* verb and goal of the other */
                                          /* instantiated verbs */
```

Figure 13. Apply in IIS-LD.

frame has three slots: Goal indicates the state attained by the verb, Action indicates the action taken by the verb, and Precondition indicates the states that must have been attained for the verb to execute its action.

Interview strategy. The nine generic attentions in SIS are sufficient to describe the interview strategies in SIS-More and IIS-LD.

Figure 9 shows how we use the Prototype descriptor to define the verb "construct" in the dictionary. Figure 10 defines Design_DB, IIS-LD's top-level task. The system first creates an initial model and converts it to the entity-relationship description. It then updates and checks the domain model until there are no more queries and attentions.

Figure 11 is a Task description of the Planning strategy associated with FoundCandidate. If this attention is generated, candidates of the verbs that can precede a given verb are returned and displayed. In Figure 12 the Def-gen-attn descriptor defines the generic attention FoundCandidate using the primitive attention SameValue. FoundCandidate is activated if there is a path linking two different nodes. The Apply descriptor in Figure 13 defines the objects on which to focus FoundCandidate: It tells the system to focus on

(1) the goal of the verbs in the dictionary (the prototype) and the precondition of a given verb in the plan structure (the domain model), and

(2) the goal of the verbs and the precondition of a given verb in the plan structure.

Dialogue. In the example dialogue between a user and IIS-LD in Figure 14, the system uses the Planning strategy. Since IIS-LD does not know the noun "part," it asks a question to locate it in the noun hierarchy. Since "part" here does not refer to a specific part, the answer is "No." IIS-LD then instantiates this noun. FoundCandidate is generated by analyzing the model, and the candidate verbs are picked up and displayed. IIS-LD guesses from the default value of the Construct verb frame that the superclass of "part" is a PhysicalObject, and the system lists the noun's possible attributes. Dialogue continues and IIS-LD outputs the designed conceptual schema at the end. This example shows that attentions are generated not only when something is improper or criticized but also when background knowledge can make the elicitation more efficient.

IIS-DB

Static analysis such as that used in More and IIS-LD is best suited for cases where the metastructure of the domain

```
i?- interview.
Please enter the information requirement.
>I want to know the names of parts which are constructed from other parts.
Is 'part' an instance of an entity ?
> No.
Do you want to know the names of part-1's which are constructed from part-2's ?.
> Yes.
Please enter the next information requirement.
> No more.
Let's discuss the following fact:
        F1 :Part-1's are constructed from part-2's.
I think you should store the information about the following verbs which can precede 'construct':
        1.buy 2.sell 3.make 4.ship 5.stock 6.supply...
Do you agree ?
> Yes.
Which do you want to store ?
> 6.
Please enter the information requirement about 'supply'.
> Some parts are supplied from other companies.
Is 'company' an instance of an entity ?
> No.
Do you want to store the information about "Part-1's are supplied from company-1's" ?
> Yes.
Let's discuss the following facts:
F1:Part-1's are constructed from part-2's.
F2:Part-2's are supplied from company-1's.
Is the next fact true ?
        F:Part-1's in F1 are constructed from part-2's in F2.
> Yes.
Let's discuss the following noun:
        N:Part.
I think it's one of the physicalObject.
Is it right ?
> Yes.
Please choose the attributes of the part from:
1.size 2.weight 3.cost 4.price 5.(other)
> 2, 3.
  .
  .
  .
Ok, I start designing conceptual schema. Wait for a moment, please.
Designed conceptual schema is as follows:
        construct (part-1,part-2)
        supply (part,company)
        part (name,weight,cost)
        company (name,address)
```

Figure 14. Dialogue with IIS-LD.

Table 3. Primitive attentions for dynamic analysis.

Primitive attention	Activation criterion
FrameInD$_1$NotInD$_2$	A frame in D_1 but not in D_2
FrameInD$_2$NotInD$_1$	A frame in D_2 but not in D_1
SlotInD$_1$NotInD$_2$	A slot in D_1 but not in D_2 for a frame in both D_1 and D_2
SlotInD$_2$NotInD$_1$	A slot in D_2 but not in D_1 for a frame in both D_1 and D_2
ValueInD$_1$NotInD$_2$	A value in D_1 but not in D_2 for a frame and a slot in both D_1 and D_2
ValueInD$_2$NotInD$_1$	A value in D_2 but not in D_1 for a frame and a slot in both D_1 and D_2

Table 4. Generic attentions for dynamic analysis.

Generic attention	Primitive	Activation criterion
FrameNotExist	FrameInD$_2$NotInD$_1$	A frame in D_2 but not in D_1
SlotNotExist	SlotInD$_2$NotInD$_1$	A slot in D_2 but not in D_1 for a frame in both D_1 and D_2
ValueNotSame	ValueInD$_2$NotInD$_1$	A value in D_2 but not in D_1 for a frame and a slot in both D_1 and D_2

knowledge is well understood. Static analysis is not as useful for the logical design of a database for several reasons.

- It is difficult to prepare sufficient question strategies in advance.
- Domain experts often are unable to answer on the spot.
- Some problems can be noticed only in actual use.

The last point was a particular problem in the database design domain. We often detected a model deficiency while using the implemented database. (Imagine the situation in which a user is asked what kind of relationship is missing without having the actual data, that is, without running the database program.)

Logical design and dynamic analysis. IIS-DB, an improved version of IIS-LD, employs dynamic analysis to overcome this difficulty. IIS-DB includes IIS-LD as one of its components. By interviewing the user, IIS-DB acquires the domain knowledge that maps the real world into the structure of the database that the domain expert/user wants to build. IIS-DB uses IIS-LD to acquire the initial knowledge and then refines it by having the user use the implemented database.

Queries raised to IIS-DB assume the latest model that the user has in mind. Deficiencies in the knowledge appear in the disagreement between the model assumed by the query and the implemented model. The system can raise questions based on this disagreement and modify the model accordingly, simulating the behavior of a database design expert. If a natural-language user interface is provided, the system can explore the cause of the disagreement without imposing an unnecessary burden on the user.

This refinement process is iterative, going through construction/problem identification/reconstruction cycles. IIS-DB evaluates the prototype database to identify problems, analyzes the problems, and corrects the domain model. It then reconstructs the database based on this corrected model and repeats the evaluation.

Applying knowledge and analyzing results are both important in dynamic analysis. For example, in the case of dynamic analysis during knowledge acquisition for a diagnostic expert system, applying knowledge means carrying out a diagnosis, and analyzing results means comparing the predicted results with the decision made by the expert.

As is clear from the above description, IIS-DB initiates an interview with the query in question as a clue, based on which modifications are made to the logical design. Once the clue is obtained, there is no substantial distinction between static and dynamic analysis. The strategies and the mechanism are the same. The same database would have been obtained if the clue obtained dynamically were known from the beginning.

Generic attentions. The purpose of dynamic analysis is to refine the initial incomplete domain model. This refinement is based on comparing two networks, the current domain model (D_1) in the knowledge base and the network representing an expert's problem solving (D_2). An input query in IIS-DB represents the partial plan structure that the user has in mind about the domain, and it can be compared with a plan structure in the knowledge base.

Since frames represent the network, there are only six primitive attentions in addition to those for static analysis (see Table 3). The two sets are similar, but their activation criteria and therefore their meanings differ.

In IIS-DB, D_1 represents the world covered by the logical design, and D_2 represents a part of the world that the user has in mind. We are interested in three cases for this discussion:

- Case 1. The intersection of D_1 and D_2 is the world that the user requests and the system can handle.
- Case 2. Those in D_1 and not in D_2 are the part covered by the logical design but not in the user's mind.
- Case 3. Those in D_2 and not in D_1 are the part the user has in mind but not covered by the logical design.

As is clear, discovering case 3 is important in IIS-DB. Errors in D_1 are generally difficult to find, and they can be detected as contradictions in case 1.

As in SIS, we can define generic attentions using these primitive attentions (see Table 4).

Task-specific attentions and their processing. Our analysis with IIS-DB has resulted in five task-specific attentions: FrameNotExist generated two, SlotNotExist generated two, and ValueNotSame generated one. IIS-DB activates and processes them in various situations. VerbNotExist, instantiated from FrameNotExist, is generated when a verb frame in the query's partial plan structure does not exist in the database's plan structure. This corresponds to the situation where the user enters a query about an activity not known to the domain model. Either there is no frame for a verb with that meaning, or a synonym was used. IIS-DB first checks the synonym possibility. If it is a valid synonym, the frame is renamed. Otherwise IIS-DB adds the verb to the plan structure.

NounNotExist, also instantiated from FrameNotExist, is generated when a noun frame in the query's plan representation does not exist in the database's plan structure. As in VerbNotExist, the user has entered either an entity unknown to the domain or a synonym of a known noun. IIS-DB interviews the user and adds a new noun frame if needed.

VerbSlotNotExist, instantiated from SlotNotExist, is generated when a slot in a verb frame in the query's partial plan structure does not exist in the corresponding verb frame in the database's plan structure. This implies that a new attribute is needed in the domain model. IIS-DB asks the user if this attribute is to be added or not. If the answer is yes, it is added to the verb frame.

Similarly, NounSlotNotSame, also instantiated from SlotNotExist, is generated when a noun frame slot in the query's partial plan structure is not among the slots of the corresponding noun frame in the database's plan structure. IIS-DB asks the user if the attribute should be included, and adds it if so.

VerbSlotValueNotSame, instantiated from ValueNotSame, is generated when a slot's value in a verb frame differs in the query's partial plan structure and the database's plan structure. There are two cases:

(1) A new relation or a modification of an existing relation is needed.

(2) Either the entity name corresponding to the value used in the query is a synonym of an entity in the plan structure, or the entity name subsumes (or is subsumed by) some entity in the plan structure. For example, "company" subsumes "factory."

IIS-DB checks if a synonym or a subsumption relation exists and, if so, replaces the slot value of the query's plan representation so that the search becomes possible. If not, the system initiates an interview and modifies an existing verb or creates a new verb frame after renaming.

Configuration. IIS-DB consists of IIS-LD and other modules, including a database-management-system interface, a knowledge base for database construction, and a supervisor (see Figure 15). The supervisor monitors each module's behavior.

IIS-DB translates natural-language queries into queries written in Data Manipulation Language. The interface module then transfers these queries to the database management system and receives the search results. It is necessary to have a database management system for IIS-DB to actually work. The present study uses a relational database management system written in Prolog that can process Data Manipulation Language of the Structured Query Language type.

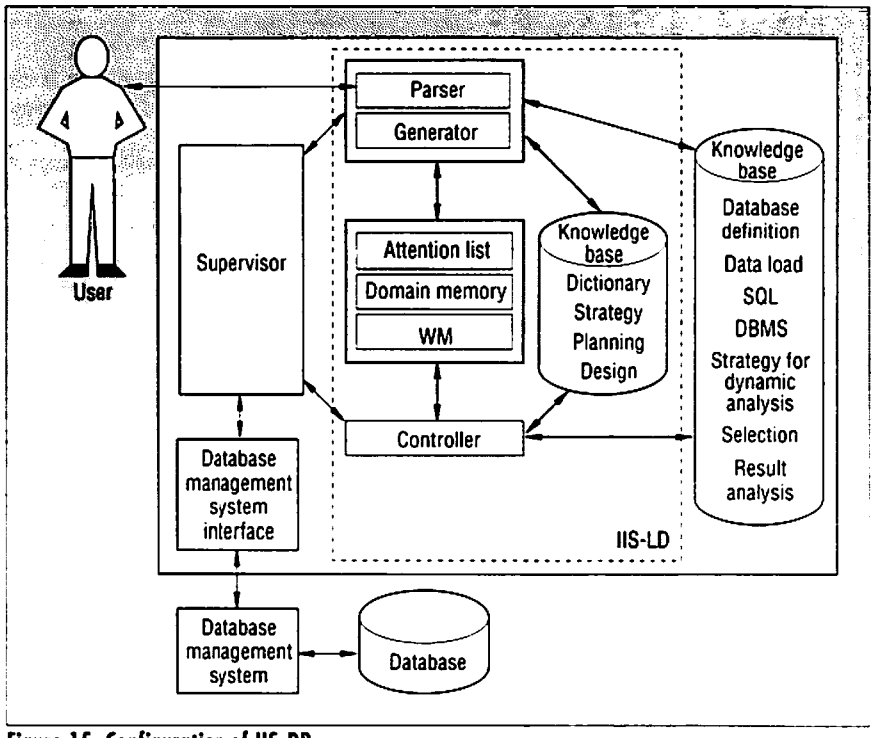


Figure 15. Configuration of IIS-DB.

Knowledge added to IIS-DB. The following knowledge is stored in the newly added IIS-DB knowledge base:

- Definition — knowledge that defines the database (using Data Definition Language) based on the result of logical design.
- Data load — knowledge used to generate the specification for the initial data format and the initial data load program.
- Query processing — knowledge to generate attentions by comparing the system's plan structure with the one estimated from query (Def-gen-attn and Apply descriptions).
- Strategy for dynamic analysis — knowledge to resolve each attention (Task descriptions).
- Structured Query Language (SQL) — knowledge to generate the query from the partial plan structure using the Data Manipulation Language.
- Selection-result analysis — knowledge to generate the response to the user from the search results (Task descriptions).
- Modification — knowledge to generate, on the basis of the modified domain model, both the specification of the format for the additional data and the database modification program.

Dialogue. An example dialogue between IIS-DB and a user is shown in Figure 16. Ten steps are delineated by alternating shaded bands. Steps 1 and 2 show the creation of the prototype database program. First, a database consisting of the relations supply, stock, and part is created. The questions raised by the system during this phase are based on IIS-LD reasoning. The response here is similar to the dialogue with IIS-LD (Figure 14). Figure 17a shows how the database's plan structure is modified during the course of interviewing in the dialogue in Figure 16. The solid rectangles in Figure 17a show the plan structure created by the dialogue in step 1, which is the domain model up to this point. This is the result of static analysis. Two verbs ("supply" and "stock") and two nouns ("part" and "warehouse") each have instances and attributes.

|?- interview.

Please enter the name of your database.

> makerDB

Let's start designing a conceptual schema of makerDB.
Please enter the information requirement.

> Find the name and the quantity of parts which are stocked.

Is 'part' an instance of an entity ?

>No.

Please enter the information requirement.

>Find the name of parts which are supplied.

Please enter the information requirement.

>None.

Let's discuss the following noun: N: part.
I think that it's one of the physicalObject. Is it right ?

>Yes.

Please choose the attributes of the part from:
1. size 2. color 3. price 4. (others) (.<nl> for none)

> 2, 3.

Ok, I start designing a conceptual schema. Wait a moment, please.
Please choose the key attributes of the part from:
1. name 2. color 3. price
Which ?

>1.

The Designed conceptual schema is the following:
type (part, price, int)
type (part, color, char)
type (part, name, char)
rel (stock, [part], [quantity])
key (stock, [part])
hie (stock, part, [],1:1)
rel (supply, [part], [])
key (supply, [part])
hie (supply, part, [],1:1)
selEnt (part, [], [name, color, price])
key (part, [name])

|?- do_format.

Ok !! I start making a load program.

|?- do_load.

Ok ! Load start !!
Completed making the first database !

|?- do_use.

Please enter a query sentence.<none. for end.>

>Find the name and the color of parts which are stocked.

select name, color from part, stock where
stock.part=part.name

name	color
nail	black
wood	blown
body	white
furniture	yellow

Please enter a query sentence. <none. for end.>

> Find the name and the quantity of parts which are supplied from marui.

Is 'marui' an instance of an entity ?

>Yes.

Please enter the entity name of 'marui'.

>Company.

Let's discuss the following noun: N: company.
I think that it's one of the organizations. Is it right ?

>Yes.

Please choose the attributes of the company from:
1. address 2. (others) (.<nl> for none)

>.

I think that you should reform the following table. Is it right ?
1: reform 'rel (supply, [part], [])' to
'rel (supply, [company, part], [quantity])'

>Yes.

Please choose the hierarchical relation of company and part of 'supply'
from: 1. 1:1 2. 1:n 3. n:1 4. m:n Which ?

> 3.

Ok !! I start making a load program.

|?- do_modify.

Ok !Load start !! Completed modifying the database !

|?- do_use.

Please enter a query sentence. <none. for end.>

> Find the name and the quantity of parts which are supplied from toyota.

Is 'toyota' an instance of a company ?

> Yes.

select quantity, part from supply where company=toyota

quantity	part
500	body
300	tire

Please enter a query sentence. <none. for end.>

> Find the name of parts which are constructed from tire.

I think you should reform the following table. Is it right ?
1: add 'rel (construct, [part(0), part (1)], [])'

> Yes.

Please choose the hierarchical relation of part(0) and part(1) of 'construct' from: 1. 1:1 2. 1:n 3. n:1 4. m:n Which ?

> 4.

Ok !! I start making a load program.

|?- do_modify

Ok ! Load start !! Completed making the first database !

?- do_use.

Please enter a query sentence. <none. for end.>

> Find the name of parts which are constructed from tire.

select part_0 from construct where part_1=tire

part_0
car

Please enter a query sentence.<none. for end.>

> None.

Figure 16. Dialogue with IIS-DB. User input is in bold type. Alternating unshaded and shaded bands separate the 10 steps.

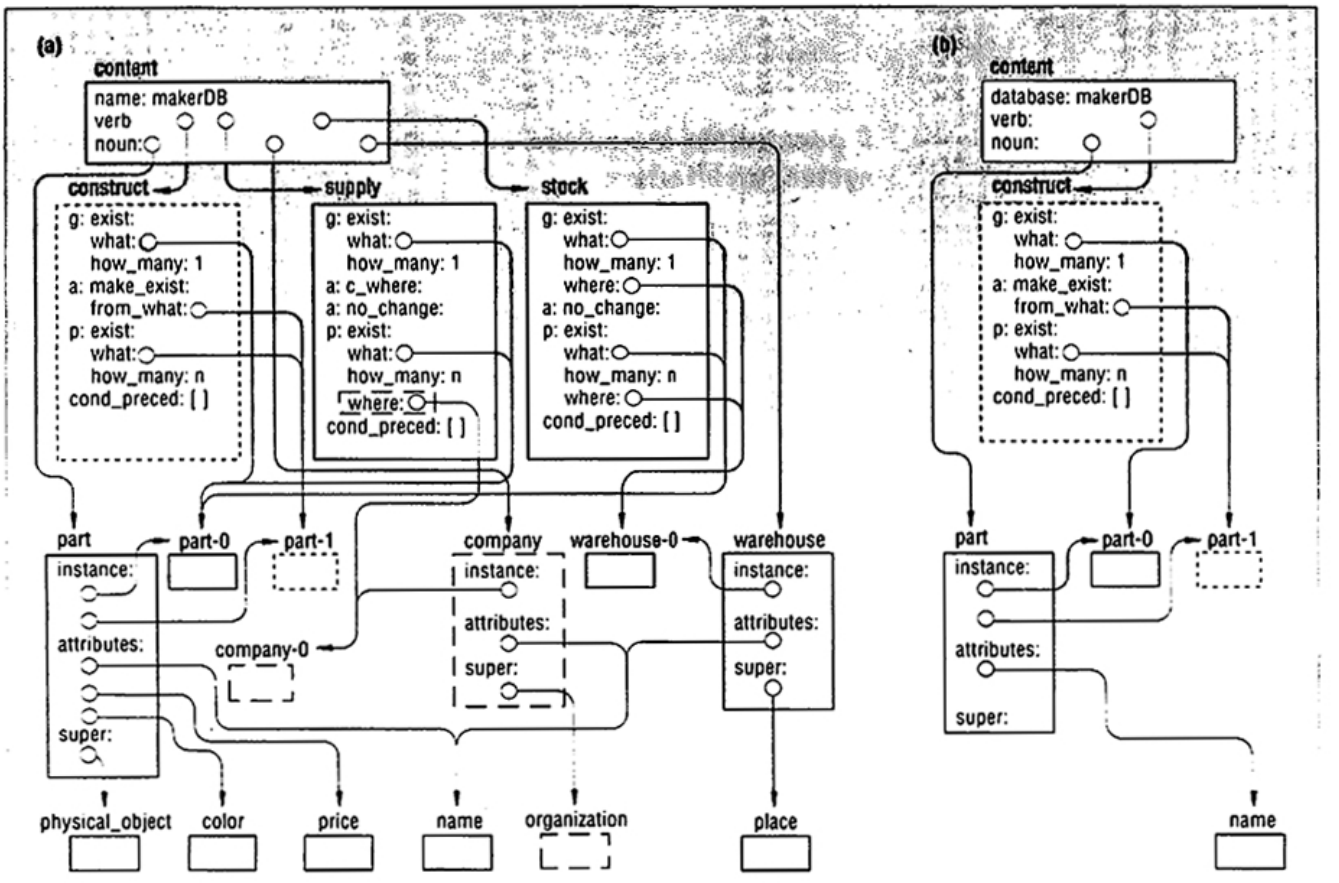


Figure 17. Modification of plan structure: (a) Plan structure through step 8 of the dialogue in Figure 16; (b) Plan representation for step 8, "Find the name of parts which are constructed from tire." Solid rectangles represent the structure through step 1. Dashed rectangles are added in step 5, and dotted rectangles are added in step 8.

Step 3 shows the creation of the prototype database program, and Step 4 shows its initial use. The domain model in the database answers the query in step 4. Steps 5-10 show the dialogue that continues because there are discrepancies between the domain model and the query. In step 5, IIS-DB does not know the term "marui" in the query, so it asks whether it is an instance of some entity. Since the answer is yes, it then asks the name of the entity. The user gives the name "company." The system then looks for the slots of the verb "supply" and finds an abstract noun, "organization," at its where-slot. It responds as if it knows that a company is one of the organizations and then asks for the attributes. Since the noun "company" appears in the query at step 5 but was not in the plan structure, the system creates the corresponding new frames and adds supply preconditions, including an exist:where-slot. The dashed rectangles in Figure 17a have now been added. The updated plan structure consists of both the solid and dashed rectangles. IIS-DB creates the new load program and adds the data at step 6 (see Figure 16). Since it now knows in step 7 that "toyota" is a company, it does not repeat the same questions, but simply confirms that this is true. No problem arises since we have added the data.

In step 8, we can see how the attention is generated from the query, "Find the name of parts which are constructed from tire." The corresponding partial plan structure of this query is shown in Figure 17b. By comparing the partial plan structure of the query and the plan structure formed up to this point, IIS-DB found no corresponding "construct" and "part-1" frames in the plan structure. This caused IIS-DB to generate two kinds of attentions, VerbNotExist and NounNotExist, and to create the new frames (step 8 in Figure 16, the dotted rectangles in Figures 17a and 17b) according to the procedures described earlier. The verb frame "construct" and a new instance of part, called "part-1," have been added.

Knowledge acquisition. In the first part of knowledge acquisition, the user must answer interview questions without using a prototype database. It is the second part, which uses the database, that distinguishes IIS-DB; that is, it acquires knowledge by dynamic analysis. Actually using the database is a good stimulus for the user, since various requirements emerge or become clarified only after use. As IIS-DB helps the user detect these requirements, the system gradually improves itself and eventually acquires the following knowledge:

- entities (nouns) and their attributes,
- relationships (verbs) and their attributes,
- links between entities and relationships,
- synonyms among entities and part_of relations,
- links between two relations, and
- synonyms for relations.

Knowledge acquisition by dynamic analysis offers several advantages. First, knowledge acquisition by static analysis can be incomplete, reducing the number of detailed questions that would have been

required to make a complete model. Also, since it is not necessary to answer all the questions while the user's knowledge is not well organized, the burden on the user is reduced considerably.

Another benefit is that problem solving stimulates the user to produce new requirements. It is effective in making ambiguous knowledge explicit by putting the user in a problem-solving environment. Also, the system can handle a requirement that the user has not even thought of at the time of static analysis.

A generalized interview-system architecture

Since the purpose of an interview is to acquire knowledge that is useful in problem solving, the fundamental challenge is to decide what kind of knowledge to ask for in what situations. Whether the interviewee is able to answer depends on how the knowledge is asked. The best way to decide which knowledge is required is to try to solve the problem. Once we analyze and acquire the necessary knowledge, we have no difficulty when we encounter similar problems the next time. The performance of the system increases gradually.

Good questions pull the interviewee into the problem-solving process. The interviewee becomes conscious about the knowledge being used. All of the existing knowledge-acquisition tools use the problem-solving function in some way, although the degree of automation and what is meant by the problem-solving function vary from case to case.

Figure 18 shows how six knowledge-acquisition tools (including ours) relate to problem solving. Teiresias, one of the earliest knowledge-acquisition support tools, works as a debugger for Mycin, a problem solver. It is invoked when Mycin makes a misjudgment. Detecting problem-solving failures, identifying and modifying wrong knowledge, and inputting new knowledge are all left to the expert. The question strategies in More confirm whether the knowledge acquired up to a certain point is enough to solve the problem.[3] IIS-LD uses a set of question strategies to predict possible troubles that might occur when the current domain model constructs a database. Mole constructs the domain model using question strategies similar to More, and then builds a diagnostic expert system and refines the domain model based on diagnostic failure.[8] ASK (which acquires strategic knowledge rather than domain knowledge) elicits justifications from the user for specific choices among actions if the user disagrees with the choice made by the problem solver, an expert system for planning workups for chest pain.[1] IIS-DB acquires the initial domain model using IIS-LD, constructs a database system, and refines the domain model based on query failure. All these tools use the common idea of acquiring knowledge when the system fails to perform a task as expected.

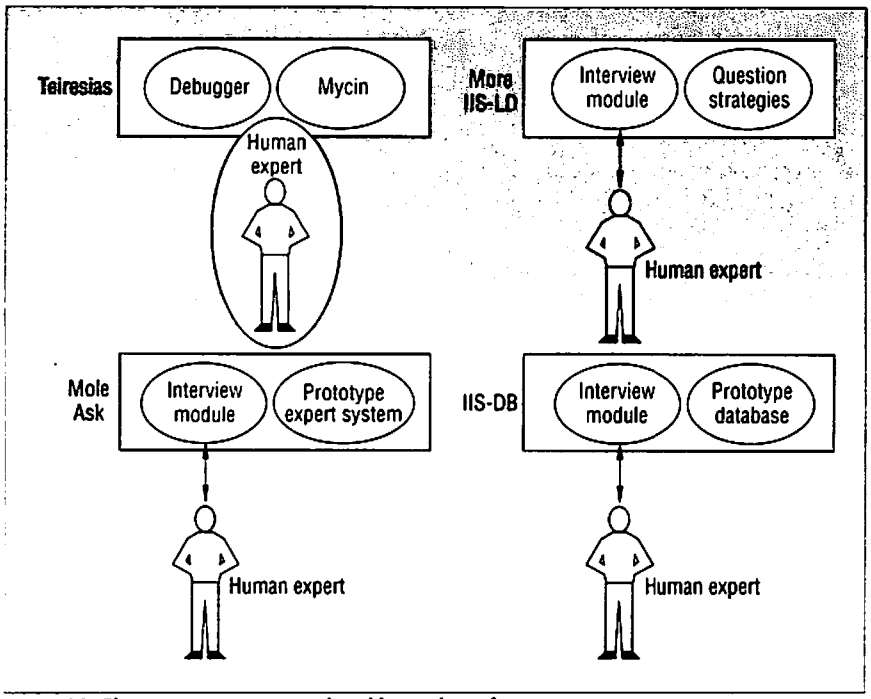


Figure 18. The interview system and problem-solving function.

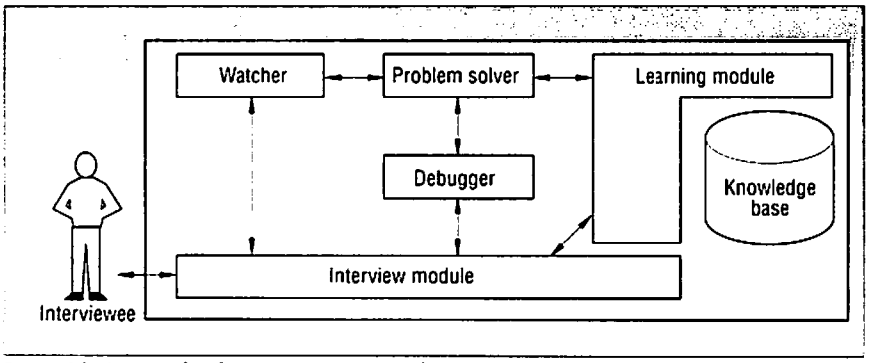


Figure 19. A generalized interview-system architecture.

Learning and interviewing. It is generally not efficient to interview while solving a problem. If the interviewer were lost in thought, it would be a very lengthy interview. We can improve efficiency by chunking completed pieces of the problem-solving process and using them for problem solving at later interviews. This essentially involves adding a learning mechanism to an interview system.

If more fundamental domain knowledge, often called deep knowledge, is available, we can use the techniques of knowledge

compilation to simplify the acquisition of task-specific domain knowledge by interview.[2] Compilation requires task knowledge; however, if instead we have available the result of using such knowledge (for example, a design example), we can compile the task-specific domain knowledge (for example, design knowledge). Since compilation is a kind of problem solving, failure to compile can initiate an interview to refine the deep knowledge and acquire heuristics that are not theoretically derivable. The Isak knowledge-acquisition tool incorporates this idea.[10]

Both learning for efficiency improvement and knowledge compilation are variations of knowledge chunking and thus are strongly related to explanation-based learning. In explanation-based learning, domain theory corresponds to the initial knowledge given to the interview system, and explanation corresponds to the problem-solving and interview processes. What is acquired is the updated domain theory.

Proposed architecture. Our proposed generalized interview-system architecture is shown in Figure 19. The problem-solver module solves a problem during the course of interview, while the watcher module monitors the problem solver's behavior. When it becomes impossible to continue, the watcher passes information about the problem-solving history and the requirements for missing knowledge to the interview module, and prompts for questions. The debugger controls the problem solver and makes it possible to solve a partial problem. The interview module controls the overall behavior of the whole system. It monitors the problem solver through the watcher and performs the interview with the interviewee. It uses the debugger to try to solve the problem, and asks questions to test and add to acquired knowledge. The learning module monitors how the knowledge is used, and it chunks the problem-solving and interview process for efficiency by making links among pieces of knowledge. It also performs knowledge compilation. The knowledge-base module stores the knowledge necessary for interviewing as well as the acquired domain knowledge. The libraries of the former knowledge (various Prototype, Def-gen-attn, Apply, Task, and Ask descriptions) become available as experience accumulates. This architecture is intended to incorporate machine-learning techniques, especially those based on the analysis of failures. This idea has been partially implemented in Isak.

This architecture is similar to the learning-system model described by Buchanan et al.[9] A major difference is that we emphasize knowledge acquisition by interviewing. The learning module in Figure 19, therefore, includes a function to make interviewing more efficient by learning how to interview from past experience, whereas Buchanan's learns new knowledge solely from given instances.

GIVEN THIS APPROACH TO acquiring knowledge through problem solving using dynamic analysis, future work must look at failure detection in problem solving, failure type classification, and failure resolution. In general, detecting failure is easy for domains where the criteria to evaluate results are well defined. Both Mole and IIS-DB have well-defined criteria: the search must be executable, and the executed results must be correct from the point of view of human experts. If the failure is easily classified and the resolution is definitely determined, knowledge acquisition by dynamic analysis becomes relatively easy. Both Mole and IIS-DB fall in this category. Generally, however, it can be difficult to analyze types of failure in advance and determine what actions to take. Nevertheless, domain-independent resolutions for failures are conceivable based on the characteristics of the problem solver. The question is whether these classifications and their corresponding resolutions are properly defined, since the form they use depends mainly on the problem solver, not on the domain. Chandrasekaran's generic-task approach[2] together with the generic-attention concept proposed here can provide a useful guide in this direction.

Another important issue is the information transfer between the problem solver and the interview module. The results of the problem solver must be delivered to the interview module either continuously or discretely, that is, only when problem solving gets stuck. In the former case, the dialogue can become complicated because each piece of incoming information is a clue to knowledge acquisition. In the latter case, the problem-solving history can become long and its analysis complicated. Therefore, knowledge acquisition systems need guidelines on how often results should be transferred. Furthermore, these systems must administer the dependency of the problem-solving process on the knowledge base such that all knowledge-base modifications are efficiently reflected back to the problem-solving process. Unfortunately, IIS-DB does not have this capability. Each time a modification is requested, everything has to be recreated according to the new specification and the load program (although IIS-DB automatically generates both). This is not efficient and needs improvement.

A common problem for current expert systems is their rapid performance degradation when applied to domains for which they were not designed (that is, their lack of robustness). IIS-DB, for example, is totally incompetent with queries that have not been taken into account at the time of design. However, in principle the problem solver can grow continuously through dynamic analysis, provided a more efficient update mechanism is employed. Smooth resumption of the problem-solving process will facilitate continuous growth.

Acknowledgment

We are grateful to Tom Gruber and B. Chandrasekaran for useful comments on the early draft of this paper.

References

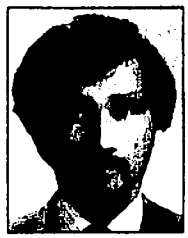
1. T. Gruber, "Learning Why by Being Told What: Interactive Acquisition of Justifications," *IEEE Expert*, Vol. 6, No. 4, Aug. 1991, pp. 65-75.
2. B. Chandrasekaran, "Models versus Rules, Deep versus Compiled, Content versus Form: Some Distinctions in Knowledge Systems Research," *IEEE Expert*, Vol. 6, No. 2, Apr. 1991, pp. 75-79.
3. G. Kahn, S. Nowlan, and J. McDermott, "Strategies for Knowledge Acquisition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 7, No. 5, 1985, pp. 511-522.
4. S. Marcus, J. McDermott, and T. Wang, "Knowledge Acquisition for Constructive Systems," *Proc. Ninth Int'l Joint Conf. Artificial Intelligence (IJCAI 85)*, Morgan Kaufmann, San Mateo, Calif., 1985, pp. 637-639.
5. M.A. Musen, "Automated Support for Building and Extending Expert Models," *Ma-*

chine Learning, Vol. 4, No. 3, Dec. 1989, pp. 347-375.

6. A. Kawaguchi et al., "SIS: A Shell for Interview Systems," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI 87)*, Morgan Kaufmann, San Mateo, Calif., 1987, pp. 359-361.
7. A. Kawaguchi et al., "An Intelligent Interview System for Conceptual Design of Database," *Proc. Seventh European Conf. Artificial Intelligence (ECAI 86)*, Pitman Publishing, London, 1986, Vol. 2, pp. 39-47.
8. L. Eshelman and J. McDermott, "Mole: A Knowledge Acquisition Tool that Uses Its Head," *Proc. Nat'l Conf. Artificial Intelligence (AAAI 86)*, MIT Press, Cambridge, Mass., 1986, pp. 950-955.
9. B.G. Buchanan et al., "Models of Learning Systems," in *Encyclopedia of Computer Science and Technology*, Vol. 11, J. Belzer, A.G. Holzman, and A. Kent, eds., Marcel Dekker, New York, 1977, pp. 24-51.
10. R. Mizoguchi, K. Matsuda, and Y. Nomura, "Isak: Interview System for Acquiring Design Knowledge — A New Architecture of Interview Systems Using Examples," *Proc. First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop (JKAW 90)*, Ohmsha, Tokyo, 1990, pp. 277-286.

Atsuo Kawaguchi is a research scientist at Hitachi's Advanced Research Laboratory. His research interests include knowledge acquisition, machine learning, operating systems, and computer architecture. He received his BS, MS, and PhD degrees in electronic engineering from the University of Osaka. He is a member of the Information Processing Society of Japan and the Japanese Society for Artificial Intelligence.

Hiroshi Motoda is a chief research scientist at Hitachi's Advanced Research Laboratory and heads its AI group. His research focuses on machine learning, knowledge acquisition, knowledge compilation, and qualitative reasoning. He received his BS, MS, and PhD in nuclear engineering from the University of Tokyo. He is an editorial board member of *IEEE Expert*, the *Journal of Knowledge Acquisition*, and the *Journal of Computer Science*, and a member of the board of trustees of the Japanese Society for Artificial Intelligence. He is also a member of the IEEE Computer Society, AAAI, JSAI, JSSST, IPSJ, and AESJ.



Riichiro Mizoguchi is a professor in the Research Department of Electronics at the Institute of Scientific and Industrial Research of Osaka University. His research interests include nonparametric data analysis, speech understanding, knowledge engineering, and intelligent tutoring systems. He received his BS, MS, and PhD from Osaka University in 1972, 1974, and 1977, respectively. He is a member of IEEE; AAAI; JSAI; IPSJ; the Institute of Electronics, Information, and Communication Engineers; the Acoustic Society of Japan; and the Japan Society for CAI.

The authors can be reached in care of Atsuo Kawaguchi, Advanced Research Laboratory, Hitachi, Ltd., Hatoyama, Saitama 350-03, Japan; e-mail, atsuo@harl.hitachi.co.jp