Porting an Extensible Natural Language Interface: a Case History

Candace E. Kalish
Matthew B. Cox
The MITRE Corporation
1 Burlington Rd.
Bedford, Massachusetts 01730
Mail Stop A040

Abstract

The KING KONG linguistic interface was developed at MITRE to be a portable natural language interface for expert systems. It is possible to port KING KONG from one expert system to another without writing more than a modest amount of code, regardless of backend architecture. We describe porting it from its original expert system backend to another expert system which was radically different in domain and representation.

I. Introduction

The KING KONG linguistic interface was developed at MITRE to be a portable natural language interface for expert systems. KING KONG has two characteristics that make it portable: it has a modular architecture, including domain-independent syntactic and morphological components, and a knowledge representation scheme which strongly adheres to the principle of declarative representation. Because of these two characteristics, it is possible to port KING KONG from one expert system to another without writing more that a modest amount of code, regardless of backend architecture, in a matter of months. In this paper we describe what makes KING KONG portable and how we ported it from its original expert system backend to another expert system which was radically different in domain and in knowledge representation.

II. Background

In 1985 MITRE began development of a natural language interface for task-oriented expert systems that would have the following properties: it would make no assumptions about either the domain or the architecture of the backend, it would require no changes in the design of the backend, it would minimize coding requirements on the programmer doing the port, and it would be easily extensible. KING KONG, the interface developed to achieve these goals, was first implemented as the front end to KRS, an air mission planning program also developed at MITRE. In 1986, KONG developers selected ISFI, an automatic programming system with a radically different architecture from KRS, to serve as the testbed backend in an experiment designed to demonstrate KONG'S portability. Within six months, one member of the group was able to port the interface to ISFI without having to write more than a small number of ISFI-specific accessing functions. KING KONG now serves as an interface to these two expert systems; we anticipate porting it to other systems in the future.

This research was sponsored by Rome Air Force Development Center under contract F19628-86-C-0001.

III. The two expert system backends

III.A. KRS

KING KONG has been used as an interface to the KRS air mission planning program for one year. The user interacts with KRS by typing to a Lisp window at the top of which is a picture of a mission template. As KRS fills in slots in response to the user's commands, the mission template on the screen is also filled in. The KRS database stores information about the domain as FRL frames; KRS plans using a generate and test algorithm in which all possible plans are generated and checked by constraints. For details about KING KONG as it runs on KRS see Zweben86.

III.B. ISFI

A programmer using the ISFI system is required to specify the constraints between the significant variables in a problem. The ISFI system then performs the problem solving necessary to derive a computation and write the corresponding computer program in a chosen target language. Lisp, C, and ADA are supported at present. A problem specification in the ISFI system consists of a network of structures representing objects and constraints, with the system relying on a knowledge base containing information about certain classes of objects and types of constraints. An ISFI user specifies a network using the available object classes and constraint types, and ISFI derives a computational path through the network by propagating along constraints. In propagating through a network, the ISFI system makes use of a number of problem solving devices, among them inheritance

of network structure stored with the object classes in the knowledge base, and application of network transformation rules. The ISFI system has so far succeeded in producing computer programs in the domains of numerical computation, graphics display and databases (see Cleveland86, Brown85).

User interactions with the ISFI system generally fall into one of two categories: specifying a problem in a way that will allow the ISFI system to derive a computation, and, less frequently, adding to ISFI's knowledge base so that it can write programs in a new domain. The ISFI system has a graphics interface which displays sections of a network and provides menu driven inspection and editing facilities. Two major problems with this interface are its inability to display a network of significant size, and its forcing the user inspecting a network to work through several levels of menu displays before finding needed information. These considerations made ISFI a good candidate for a natural language interface.

IV. Overview of KING KONG

As noted in the introduction, KING KONG has a modular architecture which aids portability. By maintaining independent syntactic, morphological, semantic, and contextual components and explicitly specifying the interactions between them, KONG allows most of the information required in porting to be specified declaratively, both speeding and easing the porting task.

IV.A. Syntax and morphology

The current morphological component of KONG employs a simple affix-stripping model. Its syntactic component is a modified Marcus parser, enhanced with strategies to handle grammatical relations. These components need not be modified during the port, since they embody information about English rather than the domain of the target backend. KONG's knowledge of linguistics means that a small amount of declarative information is enough to specify the morphosyntactic behavior of a given word.

IV.B. Semantics

Each definition identifies a word with some concept in KONG's model of the backend: either an object, which is located in a simple AKO hierarchy, or a relation, which may be located in a matrix of relation types, such as EXTENT or INTERVAL, or relation families, such as SPACE or TIME (see Bayer86). A word identified with a relation must further specify the correspondence between its semantic arguments (derived via parsing) and the arguments of the chosen relation. Relations for KRS include speed, size, operationality, range, carrying ability, etc. Relations for ISFI include scope, location, complexity and other concepts that apply to automatic programming. KONG achieves its independence from particular backend architectures by building this model of the domain and filtering all interactions with the backend through it.

IV.C. Context

Contexts are captured using data structures called scenes. A scene in the KONG interface is a stereotypical context which records the kinds of objects expected to be in prominence at a given point in an expert system user's interaction with the system, along with the user's expected action at that point. During a discourse, instances of a scene are used to record the objects mentioned in the discourse, to perform basic focus tracking for anaphora resolution and to constrain inferencing on the user's goals in the interaction. For example, as part of the KONG interface to the KRS mission planner, there is a "CHOOSE-TARGET" scene, with prominent objects (known as scene roles) being a friendly airbase, an aircraft and a hostile airbase, and with the expected action being the filling of the target airbase slot in a KRS mission template.

In order to derive actions and queries from linguistic input, KING KONG performs syntactic and semantic analysis and maps objects from the arguments and modifiers of the resulting case frame to arguments of relations and to the roles of a scene.

V. The porting task

In porting KING KONG from one backend to another, one must do the following:

1. Define scenes to represent contexts for the new domain
2. Define objects and relations for the domain
3. Define new vocabulary for the new backend
4. "Glue" the interface to the backend by writing the code that invokes backend commands or data base searches. This is the only task in porting that requires the writing of code.

Since the first three tasks involve only declarative information, it is possible to enter it via menu-driven facilities. KONG currently supports facilities for word and scene definitions; other facilities are being planned.

V.A. Scenes

Scenes model a user's interaction with the target system. The user thinks of KRS as a tool to fill in slots on a mission template, and KONG thus contains scenes for filling in each slot. The user of ISFI, on the other hand, regards ISFI as a tool for representing information about a programming problem, using the structures – nodes, constraints and so on – which ISFI makes avail-

able. The port of KONG to ISFI includes scenes modeling the user's actions in specifying nodes and constraints in a network, and in testing the ability of ISFI to write a program from the resulting network.

One must also specify the roles of a scene, the parts that various prominent objects play in the current context. In ISFI, a scene for transformation would have a role for the transformation itself, and for the network fragment matched and transformed.

Finally, one must locate the scenes in a hierarchy with respect to each other. There may be links between scenes such as parent, child, or likely successor. The degree of structure this hierarchy exhibits corresponds to the strictness of the task structure of the target system. As noted above, the ISFI user typically either builds and updates a network representing some problem, or adds to the system's knowledge base. The user's progress in these tasks will vary dramatically, depending on the difficulty of the problem to be solved and the amount of problem solving information ISFI is able to bring to bear in the domain. The structure of the scene hierarchy for ISFI is correspondingly much more flexible than that for KRS.

An example of a scene definition from the automatic programming system is:

```
(def-scene isfi-node
   :goal :fill-central-role
   :lexical-triggers isfi-node-mapping
   :inferiors (isfi-obj-class connect-to-constraint
               isfi-node-state)
   :superior in-problem-network
   :prominent-roles (obj-class constraint
                     constraint-terminal node))
```

This scene represents the context of talking about a node in the network. It may have inferior contexts in which a user talks about constraint connections to the node, the state in which the node resides, or the object class of the object inside the node. In the node context, prominent roles are likely to be constraints, object classes, the node itself, and the constraint terminals that connect nodes to constraints. It is not necessary to write this definition by hand; the menu-driven scene definition tool provides facilities for specifying all these options and produces the definition.

It is possible to perform a partial port by defining vocabulary but by not defining scenes. In fact, the initial port to ISFI was carried out this way. Users were still able to ask questions about the automatic programming system in English, but they did not have the benefits of the context and discourse tracking. This meant that they could not use pronouns or most forms of paraphrase in referring to objects, nor was KONG able to make any inferences about their goals in asking questions. The result was a "dumb" interface which processed individual sentences, but not discourse segments. As scenes were added to the system, KONG was able to make limited inferences and understand various forms of context dependent reference.

V.B. Word definitions

We will now offer two examples of word definitions and how they were performed for KRS and for ISFI. We shall show the actual code, even though some of its details may be a little obscure, because we wish to prove that KING KONG's knowledge representations are indeed declarative.

V.B.1. Actions

Here is an example of how the verb "destroy" was defined for both expert sytems. First, the morphosyntactic definition which applies to both domains:

```
(defkong destroy
   (make-word newform 'destroy
      features (copytree *VERB-DEFAULT*)
      subcategorization '(:direct-object)
      semantics (make-kernel part-of-speech 'v)))
```

KONG has extensive knowledge about syntax so the definition can take advantage of information about defaults. All one has to tell the system is that "destroy" is a verb and it subcategorizes for a direct object. All of this is accomplished through the menu-driven word definition tool.

V.B.1.a. KRS

We wish KRS to respond to an input such as "Destroy Mermin" by filling in a slot on a mission template that corresponds to the target. We specify this by defining a mapping from this sense of "destroy" to the two backend goals "fill-slot" and "change-slot." This, also, is done through a menu which contains choices for all the backend goals possible for KRS. KING KONG then writes the following code:

```
(defmapping destroy-target-mapping
   ((obj . target)
    (instr . aircraft));destroy the target with a plane
   nil
   (:fill-slot :change-slot))
```

The action to achieve this goal is associated with a particular scene, such as CHOOSE-TARGET, through another declarative definition. This action is now available for all verbs in this context, and does not need to be defined again.

```
(defbackend-action fill-slot choose-target
   :backend-goals (:fill-slot :change-slot)
   :discourse-goals (:act)
   :role-names ((mission :optional (roles))
                (target :present (clause))
                (airbase :absent (clause))
                (aircraft :absent (clause))
                (ordnance :absent (clause))
                (td :absent (clause))
                (tot :absent (clause))
                (unit :absent (clause))
                (ac-num :absent (clause))
                (pd :absent (clause))))
```

The actual execution of this backend action is one of the only aspects of the port which involves programming. A stripped down version of "fill-slot" follows:

```
(defmethod (backend-action-fill :execute) (scene)
 (send self :select-window scene nil)
 (let* ((actual-roles
          (get-roles-with-values
             (remove (send self :mission-type scene)
                     (send scene :prominent-roles))
             roles-to-use))
        (mission
          (get-role-value
             (get-role (send self :mission-type scene))
             roles-to-use))
        (backend-mission (get-backend-object mission)))
 (loop for role in actual-roles
        for slot-name =
            (get-backend-object (get-role-name role))
        and backend-role-value =
            (get-backend-object (get-role-value role))
     do (user:dump-mission-values backend-mission
          (list (cons slot-name backend-role-value)))
     finally (send scene :set-goal-filled t)
             (return :success))))
```

This code matches the arguments of the verb to the roles of the current scene, locates the actual slot item by accessing KRS's database, and puts it into the mission template. It is included only to show readers exactly what must be written by hand as part of the domain specific interface "glue."

V.B.1.b. ISFI

In ISFI, an input containing "destroy" is likely to be "destroy node x in the network." To interpret sentences like this, one defines the mapping to backend goals through a menu to produce:

```
(defmapping destroy
   isfi-node-mapping
   ((obj . node))
   nil
   (delete))
```

The action to achieve the :delete goal is declared as follows:

```
(defbackend-action backend-action-delete isfi-node
 :backend-goals (:delete)
 :discourse-goals (:act)
 :role-names ((network :present (roles))
              (node :present (clause))
             (transformation :absent (clause roles))))
```

Now one must write domain specific code for defining the action "delete:"

```
(defmethod (backend-action-delete :execute)
   (things-to-destroy)
 (loop for isfi-object in things-to-destroy
       for object-window =
           (loop for window in
                     (send (get-right-graphics-window)
                           :exposed-inferiors)
              if (eq isfi-object
                     (send window :displayed-object))
              return window)
   if object-window
   do (send object-window :erase)
      (selectq (typep isfi-object)
        (isfi:node
         (isfi:destroy isfi-object 'isfi:node))
        (isfi:constraint
         (isfi:destroy isfi-object 'isfi:constraint)))
   finally (return :success)))
```

V.B.2. Queries

The second example is the preposition "in", a word whose basic meaning applies to both the KRS and ISFI domains. We start by showing the definition for the word's morphosyntactic behavior:

```
(defkong in (make-word newform 'in
          semantics (make-kernel part-of-speech 'p)))
```

All this definition does is specify that the word is a preposition; since prepositions, unlike nouns and verbs, do not exhibit complex morphosyntactic behavior such as declension, there is no need to specify more.

Now, to add meaning to this preposition, one needs to tie it in to KONG's relational model of semantics. First, one must define a relation LOCATION with which the word will be associated. The definition for this relation is:

```
(def-db-relation location (object position)
  :type position
  :family space
  :default-relation-p t)
```

This definition says that LOCATION relates two concepts, an object and a position, and locates this relation in the type-family matrix.

V.B.2.a. KRS

Both these definitions are general across domains. But one needs domain specific code to tie the interpretation of a sentence like "the runway at Halfort" to the actual database objects it designates. To do this, one defines what is called a "relation-action" which associates a relation and its arguments with a set of object messages to send to the relation in order to extract the relevant information from the backend database. Below are several examples of such messages. The first ":country-of-airport" associates an airport's position with its home country. These messages are the only part of the port which requires programming.

```
(def-relation-action location
  '(((airport country) .
     ((position . :country-of-airport)))
    ((airport lat//long) .
     ((position . :lat-of-airport)))
    ((object lat//long) .
     ((position . :lat-of-object)))))

(defmethod (location :country-of-airport) (obj ignore)
 (car (user:mget (kong-instance-backend-object obj)
                 'user:apo)))

(defmethod (location :lat-of-airport) (obj ignore)
 (car (user:mget (kong-instance-backend-object obj)
                 'user:lat//long)))

(defmethod (location :lat-of-object) (obj ignore)
 (car (user:mget (kong-instance-backend-object obj)
                 'user:lat//long nil '(user:apo))))
```

V.B.2.b. ISFI

Here is the domain specific code for the automatic programming system's interpretation of "in".

```
(def-relation-action location
 '(((inherit node) . ((object . :inherit-event)
                      (position . :inherited-on))))

(defmethod (location :inherit-event) (ignore node)
 (mapcar #'get-accessor-from-backend-name
  (loop with node-ref = (get-referent node)
        with net = (isfi:node-network node-ref)
        for creator-record in
        (union (mapcar #'get-creation-record
                (everything-created-in-network net)))
    when (and (typep creator-record
                     'isfi:inherit-creator)
              (eq node-ref
                  (send creator-record :node)))
    collect creator-record)))

(defmethod (location :inherited-on) (to-inherit ignore)
    (send (get-referent to-inherit) :node)))
```

As with backend actions, the code for these database accesses need only be written once; as soon as they are defined, reference to them is possible through all relevant menu-driven definition facilities, and they are available for all subsequent definitions.

VI. Conclusions

Clearly, we have made some assumptions about expert systems and how they are used. We believe that in most cases, there is a limited enough number of things a user will want to do, so that one can capture the contexts he is likely to enter by defining a small number of scenes. For both KRS and ISFI this has, in fact, been true. Expert systems are usually designed to carry out a few specific tasks; if we encountered a system in which we could not identify a clearly defined, fairly small set of such tasks, we would find our scene mechanism inadequate to capture context or so bulky that it would be impossible to use. We do not believe that this is likely, but it is a good reason for believing that our system would be incapable of understanding narrative, for example. Similar reasoning applies to our use of a relational model of the backend; if we were faced with a huge array of relations including extensive overlap in the meaning of some relations we would find word definition to be prohibitively difficult.

When defining words one is still forced to think about the specific sentences in which they will be used. This is unfortunate since it introduces ad hoc, domain specific reasoning into the definition process; it also means that word definitions are rather simplistic. KING KONG has no ability to reason about word meaning in any sophisticated way, lacking the abilities of CD based systems to reason about consequence, for example. This is a serious weakness, but one that expert systems interfaces can, by and large, live with for the reasons described above. There is nothing to prevent an extension of King KONG to a richer semantics; however, we are not comfortable with the semantic models available today because they are all, even CD's, difficult to represent declaratively. We hope to have shown that the portability of KING KONG follows directly from its modularity and declarativeness.

References

Bayer, S., Kalish C. E., and Joseph, L. E. (1986) "Grammatical Relations as the Basis for Language Parsing and Text Understanding," presented IJCAI-85, August 1985 Los Angeles. Proceedings AAAI-86, pp.788-790.

Bayer, S. (1986) "A Relational Representation of Modification," Proceedings AAAI-86, pp. 1074-1077.

Brown, R. H. (1985) "Automation of Programming: The ISFI Experiments," M85-21, June 1985. Presented at the Expert Systems in Government Symposium, October 1985, McLean VA.

Cleveland, G. A. (1986) "Mechanisms in ISFI: A Technical Overview," M86-17, April 1986. Presented to the Canadian Society for Computational Studies of Intelligence-1986 Conference Proceedings, Montreal Canada.

Zweben, M., Chase, M. P. and Kalish, C. E., (1986) "Tracking Discourse & Context for an Expert System Interface," Proceedings of the Second Aerospace Applications of Artificial Intelligence Conference, 1986, pp. 200-209.