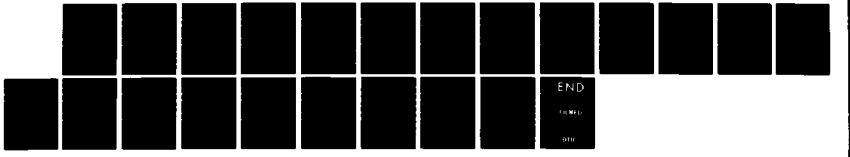
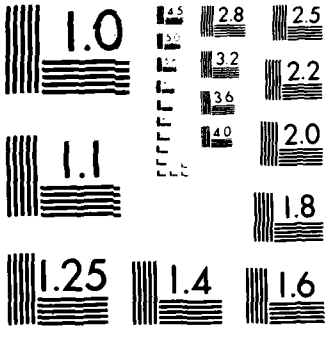
AD-A153 952 TRANSPORTABILITY IN THE CONSUL SYSTEM: MODEL MODULARITY 1/1
AND ACQUISITION(U) UNIVERSITY OF SOUTHERN CALIFORNIA
MARINA DEL REY INFORMATION S.. D WILCZYNSKI ET AL.
UNCLASSIFIED APR 85 ISI/RR-85-154 MDA903-81-C-0335 F/G 9/2 NL





ISI Research Report
RR-85-154
April 1985

David Wilczynski
Norman Sondheimer

Transportability in the Consul System: Model Modularity and Acquisition

INFORMATION SCIENCES INSTITUTE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER ISI/RR-85-154	2. GOVT ACCESSION NO. AD-A153952	3. RECIPIENT'S CATALOG NUMBER
4 TITLE *(and Subtitle)* Transportability in the Consul System: Model Modularity and Acquisition		5. TYPE OF REPORT & PERIOD COVERED Research Report 6. PERFORMING ORG. REPORT NUMBER
7 AUTHOR(s) David Wilczynski and Norman Sondheimer		8. CONTRACT OR GRANT NUMBER(s) MDA903 81 C 0335
9 PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90292-6695		10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11 CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE April 1985 13. NUMBER OF PAGES 21
14 MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document is approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

.....

18. SUPPLEMENTARY NOTES

.....

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

knowledge acquisition, knowledge bases, natural language interfaces, transportability, user interfaces

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Transportability in the Consul system means integrating new computer services within the existing framework. That integration covers a lot of ground in the Consul view of things. When a new service is built, we want to know enough about its functionality and objects so that it can share the user interface, the natural language system, and the interaction environment (error handling, data coercion, history, etc.)...

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

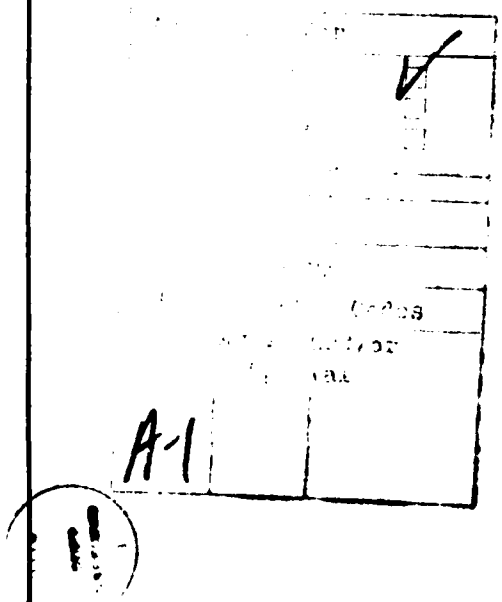
Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (continued)

We do this by building a single knowledge base that models the relevant parts of the workstation environment (and any existing services). When a new service is brought in, it is modeled and tied to the existing knowledge base through an acquisition process that guarantees the integration we demand. By acquiring a service, we have "transported" our system to it.

This paper will describe how we currently organize our knowledge base to facilitate the acquisition process and how this methodology can be extended to the natural language part of the system.



ISI Research Report
RR-85-154
April 1985

University of Southern California

David Wilczynski

Norman Sondheimer

Transportability in the Consul System: Model Modularity and Acquisition

INFORMATION SCIENCES INSTITUTE

213/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under Contract No MDA903 81 C 0335 Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U S Government or any person or agency connected with them

1. Introduction

Current thinking has it that high-performance AI systems must have intensive domain knowledge. The transportability story tells us to make these systems domain independent, that is, to separate the performance engine from the domain knowledge so that only the domain knowledge needs to be encoded in order to transport the system.

This concept must be viable since there are commercial enterprises building expert systems and natural language front-ends to data bases. However, considering the large expense to "knock off" another expert system or data base, perhaps there is more to the story.

The Consul project was chartered to investigate making interactive systems more habitable for a wide variety of workstation users, especially those with little computing experience. Through a set of prototype systems, we explored methods for allowing natural interaction--natural language requests, explanations of system facilities, user-understandable error handling--between a user and a set of online services (for text manipulation, message handling, appointment scheduling, etc.).

We faced the transportability issue by insisting that the methods we developed were (1) independent of any particular service or set of services, and (2) independent of a particular implementation of any service. These requirements are necessary considering that all interactive systems seem to evolve by incorporating new services and replacing or modifying old ones. Furthermore, adding a new service is not just a question of transporting our methods to it, but rather fully integrating it into the existing environment.

That integration covers a lot of ground in the Consul view of things. When a new service is brought into the Consul framework, we want to know enough about of its functionality and objects so that it can share the user interface, the natural language system, and the interaction environment (error handling, data coercion, history, etc.).

We do this by building a single knowledge base that models the relevant parts of the workstation environment (and any existing services). When a new service is brought in, it is modeled and tied to the existing knowledge base through an acquisition process [Wilczynski 81] that guarantees the integration we demand. By acquiring a service, we have "transported" our system to it.

This report will describe how we currently organize our knowledge base to facilitate the acquisition process and how this methodology can be extended to the natural language part of the system.

2. The Consul Models - Separating Theories

The interactions between a user, a powerful workstation, and the workstation's services are complex. Consider a user "selecting" (by whatever means) two screen positions and a MOVE menu item to effect a "Move this to there" action. Given what was pointed at and the context of the command, the user could be trying any of the following:

1. Moving a window (which was covering something he now wants to see, for example).
2. Deleting a file (if the "this" was a file and the "there" was a garbage can icon, as in the Apple LISA system).
3. Copying a user name from a memo into an address field of a message.

Obviously, some methodology is needed to scope the selections and to interpret the intent of the user. Doing this takes into account the display medium, the functional capabilities of the system, the sophistication of the user, the context of the request, etc.

Managing all this from within some traditional integrated software paradigm is problematical, especially if you believe in an evolving world where everything is changing. Instead of doing a lot of hard work and producing a well-designed, monolithic system, we do a lot of hard work and produce a well-designed modular knowledge base. That knowledge base encodes several theories about the workstation, each implemented as a piece of the overall model. The idea is as follows:

1. Each theory and its resulting model is complete unto itself and can be developed independently.
2. Each model is then specialized by actual instances from the real world.
3. The models are linked together by an acquisition procedure to effect actions consistent with each model.
4. Changing a model requires examination of the linkages made by acquisition.

This process and its result are depicted in Figure 2-1. The next step is to look at three of the models (the fourth, the atural language model is discussed in Section 4.1).

2.1. The Domain Model - How the World Works

The domain model represents the real-world activities that are relevant to the end user. The current domain model contains information about objects (messages, calendars, documents, persons, addresses, and phone numbers), inheritance (message are communication objects, calendars are time-sequenced objects), actions (messages can be sent, meetings can be scheduled), events (task completion, start up, errors, alarms), inferences (sending an indefinite message can be redescribed as a create action; a "tell me about" action can be redescribed as a display action), and relations (an employee is a person who stands in an employment relation to an organization). All of this is encoded in NIKL, a new implementation of the KL-ONE semantic network system [Brachman 78].

Since the Consul system deals with using interactive computers, the domain model also contains concepts about computers, computer users, computer objects, and computer actions in a way that typifies our overall philosophy about the domain model: all people bring some common ideas to the workstation about how things happen, how certain tasks are accomplished, what a computer s general capabilities are, etc. In fact, the entire modeling task is bounded by computer activities, and the programs that carry them out. That raises the question of how new programs (or services) are added to our system.

The main enterprise in adding a new service to our system is building the corresponding domain model concepts. That means defining the relevant concepts, putting them into the existing taxonomy, discovering and representing commonalities, introducing new abstractions, etc., etc., etc.

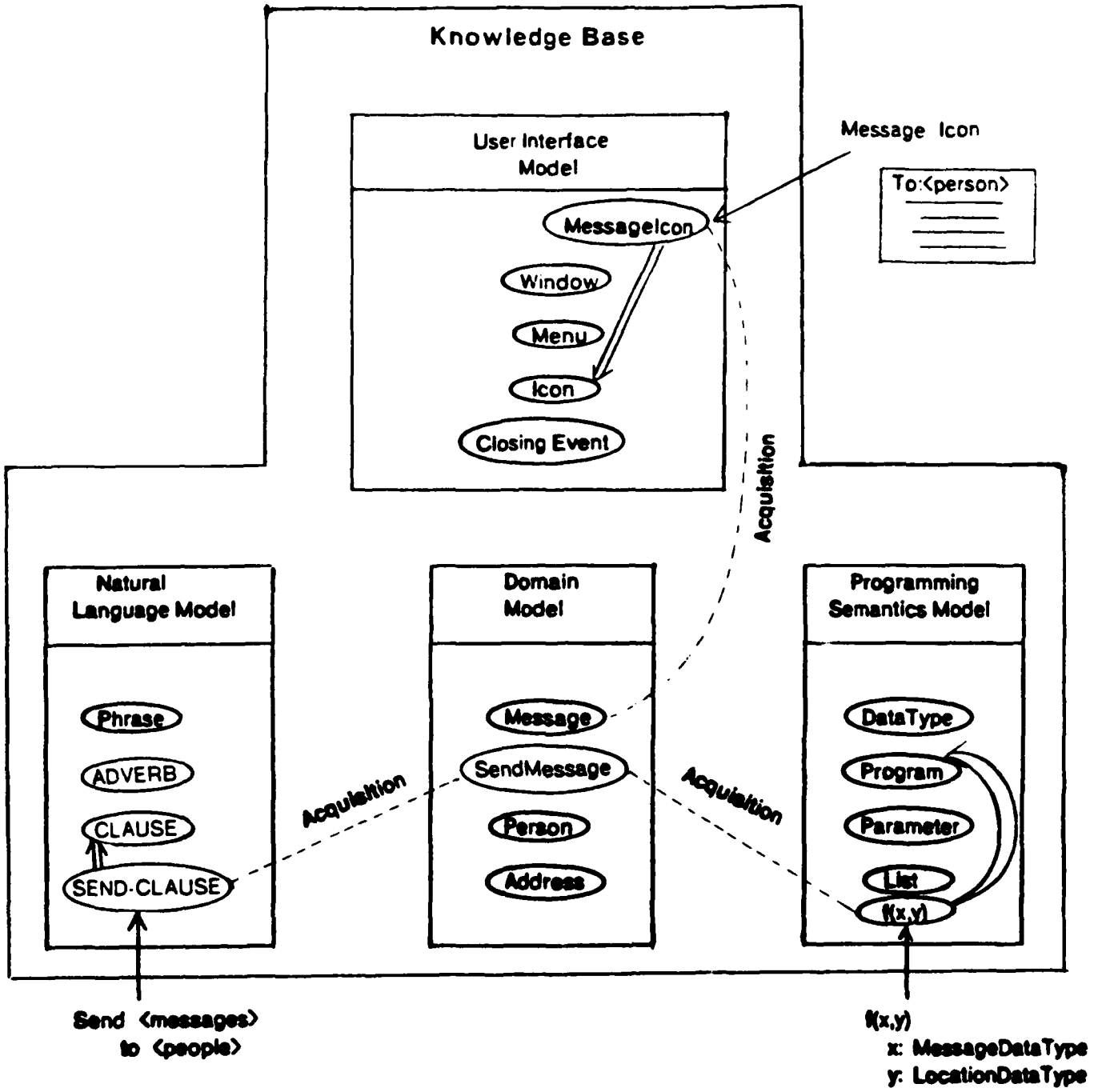


Figure 2-1: The Consul knowledge base

For example, if our system knows about messaging and calendar services, adding project planning and monitoring services involves modeling schedules, milestones, budgets, etc.; meetings might be planned at milestones, messages might be sent at key identifiable moments, and so on. The point is that there are interactions between existing services and the new one. The domain model has to be augmented with the new concepts and the old concepts may need some "updating" to properly reflect these new, perhaps unanticipated, interactions.

This exercise is neither automatic nor easy to characterize. It represents the same effort that people must make when learning about something new. What we do to make our task more manageable is to separate the characteristics of the domain from the corresponding computer programs and computer objects.

2.2. The Programming Semantics Model - How Programs Work

While the domain model is targeted for the end user, the programming semantic model is for the programmer. The concepts in this part of the model are about programs and data structures: programs have parameters and error messages, lists and arrays are kinds of composite objects, cartesian products are composed of attribute-value pairs, etc. The model tells us that all data structures must have creators and destroyers, certain data structures have special additional operators (stacks have push and pop, lists have car and cdr), and so on.

This model, disjoint from the domain model, forms a taxonomy of its own and is subject to the same inheritance properties found in all KL-ONE networks. When a new program is defined by the programmer, a KL-ONE description for it is automatically[1] built in terms of the programming semantics model--once it is modeled, we know how to call it because we know what types of parameters it takes, what defaults are involved, what effects it produces, etc. When new data structures are defined, we want to know about their type, their associated operators, their default values, and so on.

The key here is that the *names* of the programmer-defined programs and data structures are not used to define their semantics. That meaning comes from linking the newly created programming semantic description to the domain model through acquisition, as we will show later.

2.3. The User Interface Model - How Things Are Presented

The user interface is the bridge between a user and the computer's functionality. Its concerns are in many ways separate from the work to be done (domain model) and how to do it (programming semantics model). Instead, as the vehicle that gets a user from intentions to actions, the user interface model contains objects like windows, menus, pointing devices, icons; actions like selections, and opening and closing events; relationships like being on display, having screen manifestations; and general monitoring activities related to history, undo, redo, etc.

---

[1] This is essentially a simple parsing task from a programming specification to NIKL. As we shall see, the corresponding natural language case is more complicated.

We expect to be able to represent any particular user interface theory in this model. That is, any given set of interface capabilities can be embedded into the abstract model (completely analogous to the relation between a real program and the programming semantics model), the interface can be configured around its basic capabilities to act in designable ways, and finally, through the acquisition mechanism, the interface can enact certain policies by tying them to the domain model.

For example, in the user interface model we have the notion of a closing event, while in the domain model, certain events can be classified as completed tasks. Let's assume a particular interface that has multi-window capabilities. One part of someone's user interface theory may be that "all completed tasks should be closing events implemented by closing the task window, opening the completed tasks window, and putting into it a reference to the completed task."

Notice that there are three parts to establishing this policy:

1. The actual window operations must be described in the user interface model.
2. The particular closing event implementation must be described in the programming semantics model and then linked to the closing event (by acquisition).
3. That particular closing event is tied to the domain model (by acquisition).

The result is a modularity that lets the system builder:

- enforce a policy for all completion tasks, thus giving consistency to the interface, i.e., every domain action that is a kind of task completion will result in a call to the same closing event interface functions;
- change the policy by relinking the domain model's task completion concept to something else in the user interface model;
- change the actions of a closing event without changing the policy, e.g., you may want to create task completion icons instead of entries in a window;
- adapt the closing event to a different set of terminal capabilities, i.e., change the screen manifestations of the actions, not the actions or the policy, in order to account for enhanced or diminished capabilities on different hardware.

The last point makes it easier to port our system to differing types of workstations. To reiterate, we separate the intent of the policy from its actions, which are in turn separated from its screen manifestation.

3. Acquisition - Bringing Theories Together

We have gone to great lengths to build a model that addresses four separate concerns in the interactive system world: the domain model for the end user, the programming semantics model for the programmer, the user interface model for the system designer, and the natural language model for the computational linguist. The resulting modularity is our expression that these models are *definitionally* (from a KL-ONE point of view) disjoint. But definitional (i.e., subsumption) relationships are just one of many that we need to express.

There are also inferential or redescriptive linkages between knowledge base concepts [Mark 81], extensional links between knowledge base descriptions and real-world objects [Mark 82], and representation links that connect the different models. The latter are built by the acquisition process.

The current acquisition system is used when a new data structure or program is built. Its acquisition is a two-step process: first, it is defined within this programming semantic model; next, it must be integrated with the domain model by building representation links, i.e., links that tell us what programs and data structures to use given some semantic description of an action or object. As we will see, the process is contagious; the definition of a data structure will bring with it a variety of requirements for other functions and data structures. An example will illustrate this process.

3.1. An Acquisition Example

In this example, the programmer is defining a new data type called BasicMessage.Alpha. The data type definition is automatically parsed into a NIKL form as shown in Figure 3-1. The figure tells us that a BasicMessage.Alpha is a kind of Cartesian Product with five constituents: a datesent.alpha filled by Date.CUE, a subject.alpha filled by Text.CUE, a body.alpha also filled by Text.CUE, a sender.alpha filled by an Addressee.Alpha, and an addressee.alpha which is filled by an AddresseeSequence.Alpha[2].

It is important to realize that a BasicMessage.Alpha is *not* a Message; it is a ProgramObject with constituents of arbitrary names whose fillers are other ProgramObjects. What we want to say is that a BasicMessage.Alpha represents, i.e., is an implementation of, a message. That happens next.

Using browsing and explanation [Lipkis 81] tools, the programmer finds the concept that he decides his data structure represents, in this case, Message. The domain model Message tells us that a Message is a Communication Object with from, to, subject, and body roles; it can be sent, created, etc. This abbreviated figure doesn't show a lot of the other domain knowledge that is known about messages. For example, the model has "Request about object" concepts; furthermore, the model has inferential knowledge that if an object has a subject, then requests about this object can be fulfilled by telling the user about its subject. So, in requests like, "Tell me about this message," or "What is that message about?" the Consul system will know to use the subject in its answer.

Figure 3-2 is the display the programmer uses to make the linkage between his BasicMessage.Alpha and the domain concept, Message. The basic idea is to point to a role of the domain concept, a constituent of the cartesian product, and then button the Associate menu item. The process ends when all the relevant associations are made. There are many special cases. The scenario for this example highlights several of them.

1. The programmer buttons "subject," "subject.alpha," and then Associate. Since the fillers of the subject role and subject.alpha constituent, ComputerText and Text.CUE, already represent one another--indicated by the dotted line connection in Figure 3-1--the association is valid.
2. The programmer buttons "body," "body.alpha," and then Associate. The association is valid for the same reason as above.
3. The programmer buttons "from," "sender.alpha," and then Associate. The system notices that ComputerAddress and Addressee.Alpha, the respective fillers, do not yet represent each other. The programmer is asked if he wants to recursively acquire

---

[2]The Alpha and CUE suffixes are just part of a naming convention

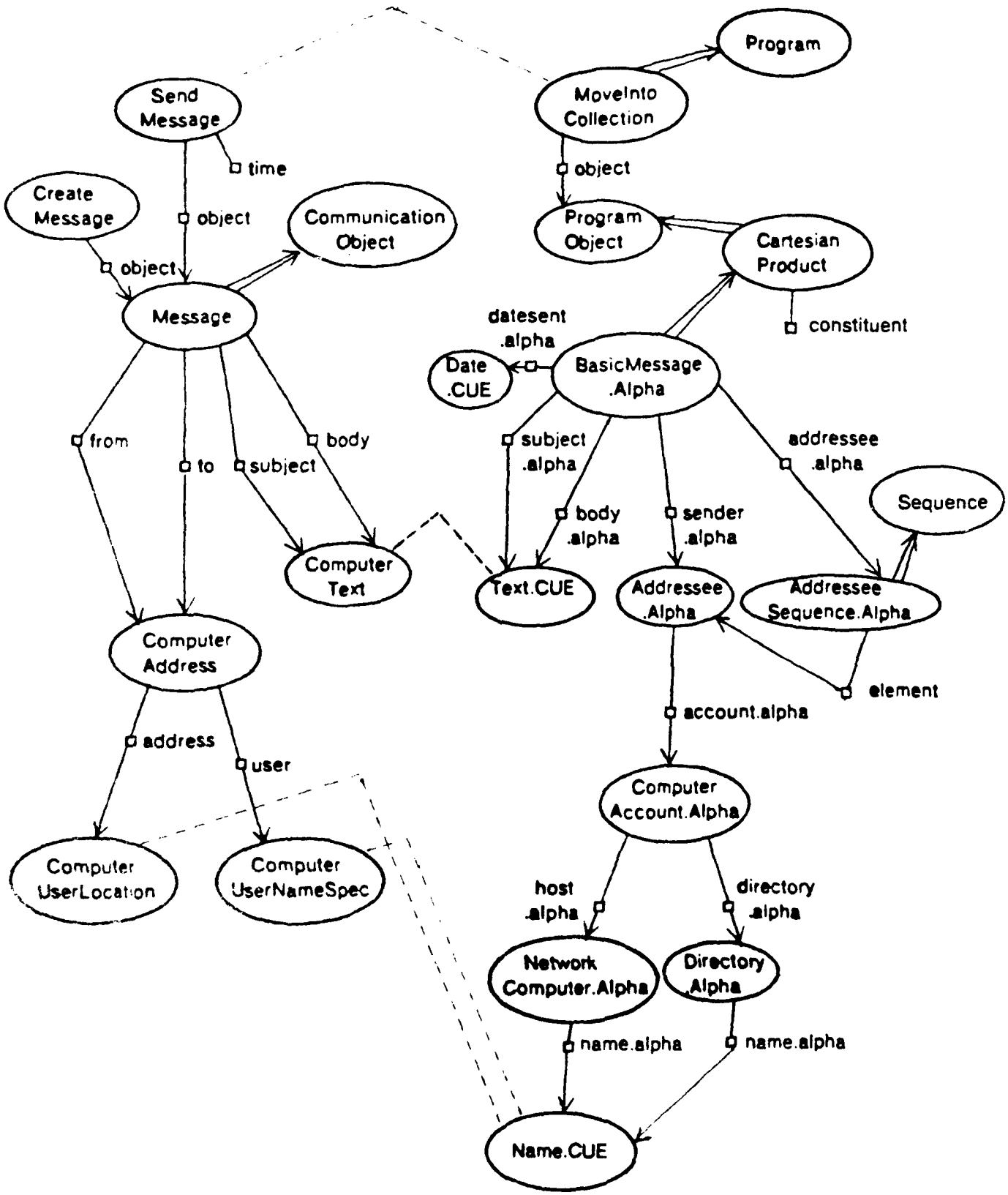


Figure 3-1: Acquiring BasicMessage.Alpha

Pertinent Roles of Message	
Concept **Message**	
part	**ComputerInformation**
responsestatus	**MessageResponseStatus**
inspectionstatus	**MessageInspectionStatus**
deletion	**MessageDeletionStatus**
currency	**MessageCurrencyStatus**
status	**MessageStatus**
sequencenumber	**Number**
subject	**Subject**
body	**ComputerText**
to	**ComputerAddress**
from	**ComputerAddress**

Definition of Cartesian Product BasicMessage.Alpha	
Type **Basic Message.Alpha**	
datesent.alpha	**Date.CUE**
body.alpha	**Text.CUE**
subject.alpha	**Text.CUE**
addressee.alpha	**Addressee.Sequence.Alpha**
sender.alpha	**Addressee.Alpha**

Acquire Commands
Associate
Back
Done
Explain
ExamineType
Help
Quit

Figure 3-2: The acquisition display

Addressee.Alpha as a ComputerAddress. He says yes. The screen changes to show the pertinent roles of ComputerAddress and the constituents of Addressee.Alpha (we will refer to Figure 3-1 for the associations).

a. The programmer realizes that the user role of ComputerAddress is to be represented by the name.alpha constituent of Directory.Alpha, and that to effect this association, he must walk down the data type through the appropriate constituents until that name.alpha constituent is visible. He starts by buttoning the account.alpha constituent, then ExamineType. This sequence of actions overlays a window with the type definition of ComputerAccount.Alpha. He then buttons "directory.alpha" and ExamineType, which overlays another window with the definition of Directory.Alpha. Now the name.alpha constituent is visible. Finally, he buttons "user," "name.alpha," and Associate. Since ComputerUserNameSpec is already associated with Name.CUE, the association between the ComputerAddress' user role and the constituent chain (account.alpha, directory.alpha, name.alpha) is valid. This more complex association is needed because the two models will have different structures. This part of the scenario emphasizes why an Addressee.Alpha does not specialize ComputerAddress in the strict KL-ONE sense and why a different kind of linkage is needed.

b. In a similar manner the programmer makes an association between address and the constituent chain (account.alpha, host.alpha, name.alpha).

c. The programmer buttons Done, thus completing the acquisition of Addressee.Alpha as a ComputerAddress. The windows are cleared, popping back to the display of Figure 3-2.

Now that Address.Alpha and ComputerAddress represent each other, the association between from and sender.alpha is valid.

4. The programmer buttons "to," "addressee.alpha," then Associate. The system sees that ComputerAddress and AddresseeSequence.Alpha don't represent each other, but notices that the latter is a Sequence whose element is an Addressee.Alpha, which does represent a ComputerAddress. Thus, the reasoning that a set of ComputerAddresses can be realized as a sequence of ComputerAccounts.Alpha makes the association valid.

5. The final association to be made is between datesent.alpha and something in the domain model. Notice that nothing *at* Message does the job. What we will want to do is make some association between the time of the SendMessage function the datesent.alpha constituent of BasicMessage.Alpha. Though unimplemented as yet, we expect it to go as follows.

   The programmer buttons Message and Help and gets a table of concepts that have Message as a filler of some role. He sees SendMessage (refer to Figure 3-1), buttons it and ExamineConcept to see its pertinent roles. He buttons the "time" role, "datesent.alpha," and then Associate. The system now reasons that for the association to be valid, there must be some real program in the programming semantics model that implements the SendMessage function. It finds the association with MoveIntoCollection, looks for a specialization with BasicMessage.Alpha as its object. It finds SendAlphaMessage (see Figure 3-3). The programmer then associates the time role with the dateofoperation which he in turn declares to be co-referential with the datesent.alpha constituent. Figure 3-3 shows the result of all this.

6. The programmer now buttons Done and BasicMessage.Alpha has been acquired.

Notice that not all the roles of Message were realized in its implementation as BasicMessage.Alpha. Some, like sequencenumber, may have a functional association in a way dual to datesent.alpha's functional association. Others may simply have no realization. Similarly, the programmer may have had a constituent like priority.alpha that had no counterpart in the domain model.

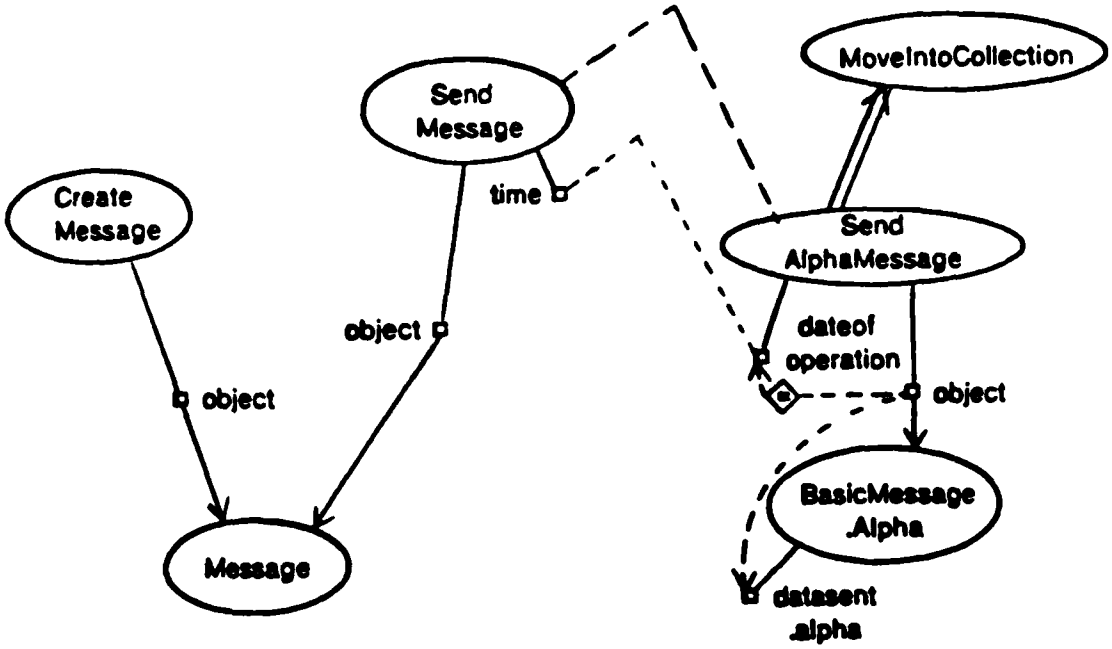


Figure 3-3: The datesent.alpha association

3.2. Model Mismatches - Differences in Subtlety

Since the various models will probably evolve independently, there are bound to be mismatches in their expressiveness. This situation is normal in human communication. One speaker is often able to express some thought with more subtlety than the listener can grasp; if the details are critical to the dialogue, there is a problem. Think about talking to children.

In our message example, our system may be prepared to entertain questions about a message's status; however, if there is no realization in the programming part, the best answer the system can give us, "I know about message status, but the implementation doesn't have a counterpart."

The dual case in which the programmer supplies more function or data than is modeled in the domain can also be handled. Certainly, the domain model can be built to account for the extra capabilities, but if it isn't, the system can deal with it in syntactic terms.

For example, assume BasicMessage.Alpha had a priority constituent whose values ranged from 1 (low priority) to 10 (high priority). If object importance were part of the domain model, it would be

linked during acquisition to this priority constituent so that user requests like, "Send this message to Jones, now!" or "Do I have any important messages?" would interact appropriately with it. If object importance is not modeled, then the same request might have to be, "Send this message to Jones [priority = 9]" or "Do I have any messages [priority > 7]?". Not graceful perhaps, but at least there's an answer.

Knowing how to handle model mismatches is important, but even more critical is knowing what to do when the model changes. In our framework, we have some idea about the implications of such changes. So, for example, if a datatype changes, it must be reacquired, or if new domain model is added, we must check to see if it covers some existing functionality not yet accounted for. All of this applies to the natural language part of the system as well.

4. The Consul Natural Language Interface

4.1. The Consul Natural Language Model

Just as there is a model for other aspects of the interactive environment, there is a model of how natural language is organized to describe the actions, events, relations, and objects seen within the environment. There are two parts to the model syntactic and semantic.

The syntactic model is essentially a domain-independent dictionary and grammar [Bobrow 78]. A parser uses this information to analyze the constituent structure of input sentences and perform some normalization, e.g., allowing for underlying subject and object in the face of passivization and showing the basic propositional content separately from the illocutionary force.

The semantic model is in the form of a theory of the types of word and phrase senses and their relation to the entities described in the other models of the Consul system [Sondheimer 84a]. Using NIKL, each distinct word sense or phrase type is shown as a separate concept. Each of the phrasal concepts has role restriction associated with it that shows the semantic categories of the immediate constituents of the phrase types. To aid in the task of specifying these selectional restrictions, among other things, "abstract" phrase types and classes of word senses are described by concepts. Number restrictions show optionality/required marking.

Concepts for both word and phrase types are connected to the other models of the system through data associated with concepts to show basic translations and translation rules for individual role restrictions [Sondheimer 84b]. These translation rules can be conditional, e.g., testing whether a syntactic relation is realized in a phrase. The rules make the connections between the natural language model and the other models.

Figure 4-1 shows a concept for recognizing a variety of clause describing the sending of a message. Figure 4-2 shows (part of) one meaning representation for an example sentence involving the sending of a message. The attached data labeled TRANSLATION and TRANSLATION RULE give the linkages that support the translation process.

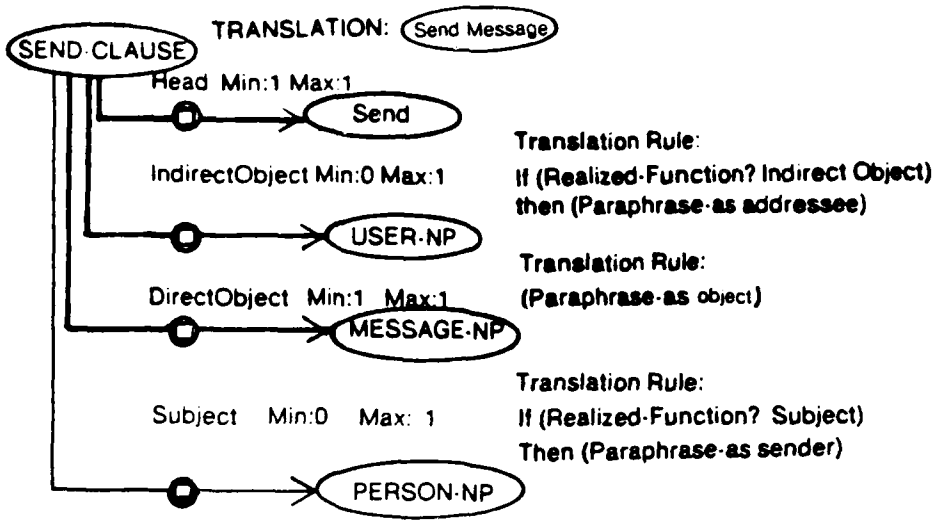


Figure 4-1: Concept for sending clause

4.2. Tools for Expanding the Natural Language Theory

Extension of the natural language interface to support access to new types and levels of services involves adding new concepts to the natural language model and new linkages from those concepts to the rest of the Consul models.[3] The concepts allow for the natural language descriptions of new options, requests, objects, etc., involved in the new service. The linkages support the mapping process from the natural language to actions.

The extension process begins with the developer knowing how functionality of the service is described in the domain and other models, and what a user will say to access that functionality. The developer begins by exploring how these natural language forms differ from already modeled structures. He considers the structure of the new phrases, the restrictions on their constituents, and the restrictions on the use of the new phrases and words. Important in this process are the abstract phrase types. Nearly always, the new phrases can be seen as specializations.

As the result of his exploration, the developer takes part in a set of NIKL activities to expand the natural language model. He specializes new concepts and roles to define new word classes, word senses, phrase types, and new semantic relationships, and he builds role restrictions to describe the structure of phrases. In Figure 4-3, we show the additions the developer would make to add priority status through adverbial reference to descriptions of sending. The assumption is that the ADVERB, CLAUSE, and SEND-CLAUSE concepts already exist. A new class of transmission priority adverbs

---

[3] The syntactic component does require acquisition of newly significant words and idioms. An interactive facility is available for this purpose

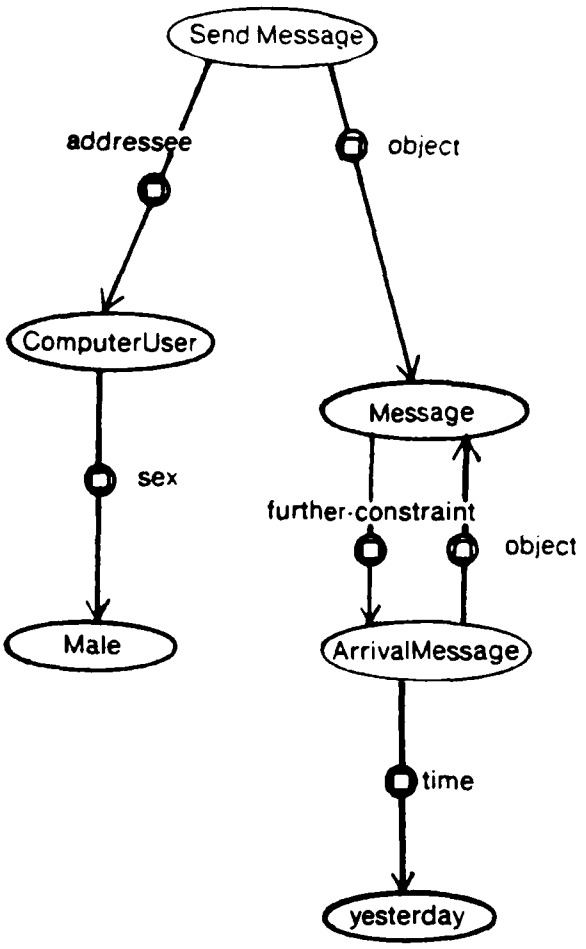


Figure 4-2: Partial meaning representation for "Send him the message that arrived yesterday"

would be defined as a specialization of the class of adverbs. Word senses that are members of that class would be constructed. A role that distinguishes adverbials showing transmission priority modification from other uses of adverbial modification would be added and restricted at the concept for sending clauses.

In each of these areas, we have built or designed facilities to aid the developer. Exploration of the network proceeds through our browsing mechanism, which displays a concept specialization hierarchy. It provides access to a program to graphically display a concept, an English language explanation system, and an "inspector" package that presents a structured view of the internal form of a concept. These are all in operation.

Tools to enter the model are under development. Specialization of lexical categories will begin by selection of a class of the browser. Entry of new categories and word senses will then be based on completing a form. Fields taking concepts will be filled by either entering the concept's name or identifying it through the browsing system. We will also support menu selection for the completion of fields taking limited choices. Entry of new roles will be done similarly.

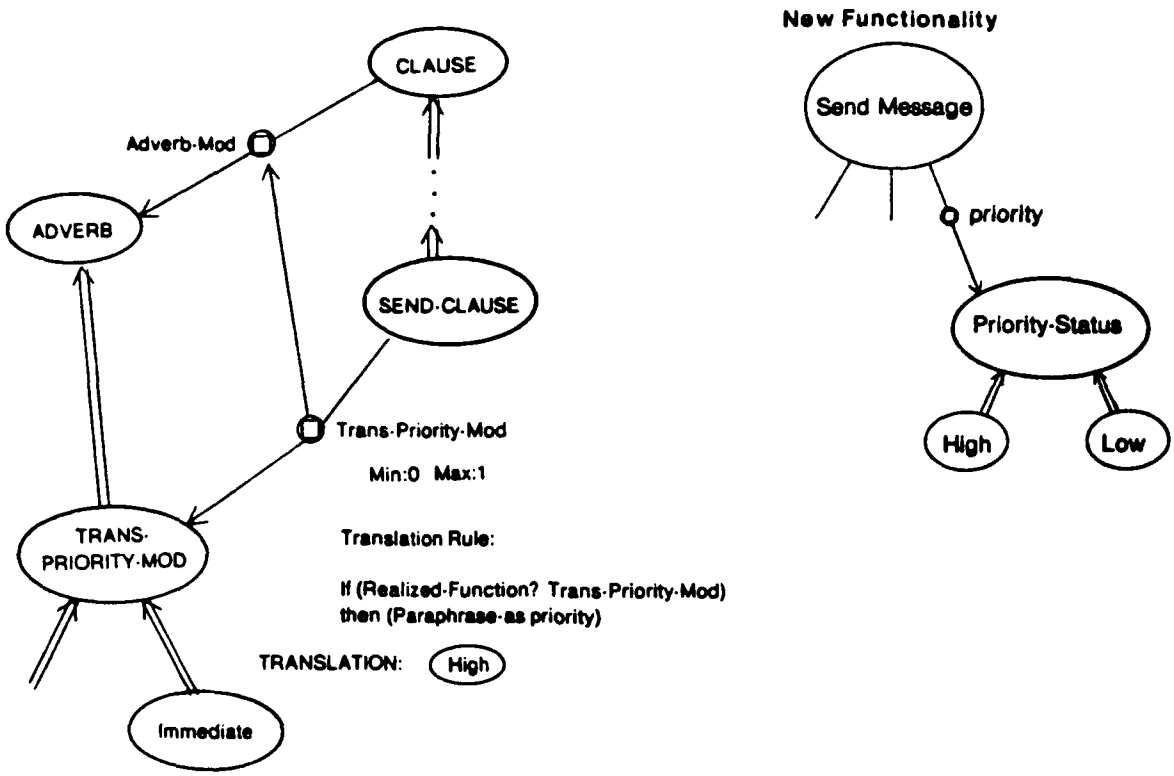


Figure 4-3: *New functionality and natural language model*

Entry of new phrasal descriptions will be somewhat more complex. It will begin with selection of a concept or concepts to specialize through the browser. We may then produce a form based on the inspector, showing the structure of the concept as it would be established through inheritance. This could be edited. Alternatively, the concept could be displayed visually. We are experimenting with a display based editing system.

4.3. Acquisition of Natural Language Model

At the same time the developer is exploring the place of the new natural language phrases in the theory, he is exploring the linkages that must be made to the domain, user interface, and programming semantics models. These must become translations and translation rules. This process is what we have been calling acquisition. The Consul paradigm supports the construction of facilities to simplify this process, just as with the acquisition situations.

In our design, natural language acquisition proceeds in the same way as other acquisition. With new concepts in the natural language model, the developer is asked to identify a concept in the other models that it describes. This becomes the translation. In the example from Figure 4-3, this would be how the translation is associated with the word sense for "immediate."

If the concept describes a phrase, the developer is asked to establish the connections between its constituents and the components of the basic translation. Many of these can be handled exactly as the connection established by chains of roles shown in the earlier existing system and a similar reference to attached data. Again referring to Figure 4-3, the domain model for priority is shown on the right of the diagram. This model allows a simple connection that relates the Trans-Priority-Mod role restriction of SEND-CLAUSE to the priority role of SendMessage. It can be supported with the same pointing or button style operations shown above.

Some of the connections established during acquisition do involve some construction of new models or reference to constants. At the moment, we assume that will require direct text entry of translation rules. However, these are clearly the minority of cases, and they tend to be used for general phenomena that we hope the developer will already find in the model (e.g., relative clause modification or question translation). Conditionality, the only remaining aspect of acquisition, can be supported with simple menu choice.

Finally, our experience has shown us that acquisition is a major occasion for uncovering model mismatch. Many mismatches arise because no stipulation has been made for natural language access to functionality, or because the natural development of the natural language model has provided (through inheritance) an interpretation of an expression that has no counterpart in other models. The formalization of the acquisition process brings these conditions to the surface. The process of connecting the models eases the task of noting where functionality has not been reached or noting where there is no functionality to reach. In fact, acquisition of new natural language models has been one of the major occasions for expanding the other Consul models.

5. Conclusions

Our modular knowledge base design faces the transportability issue squarely. To us, moving to a new domain is a knowledge base maintenance task, involving at least the following objectives:

- building a new domain model and assimilating it with the existing knowledge base;
- bringing up new functionality by using acquisition to link it to the domain model;
- updating the lexicon and natural language theory to account for new words and structure and then using acquisition.

Maintaining a knowledge base is certainly not easy, especially when it gets large; our tools are still primitive. But we think our knowledge base is transparent and easier to understand than similar systems based on rules or procedural encoding. Having this body of knowledge represented separately and explicitly allows us to build a variety of interpreters, explainers, and inference engines to work on it. Programs come and go, but a sophisticated knowledge base has a lasting property.

References

[Bobrow 78] Bobrow, R. J., "The RUS System." in B. L. Webber and R. J. Bobrow (eds.), *Research in Natural Language Understanding*, Bolt Beranek and Newman, Inc., BBN Technical Report 3878, 1978.

[Brachman 78] Brachman, R., *A Structural Paradigm for Representing Knowledge*, Bolt Beranek and Newman, Inc., BBN Technical Report, , 1978.

[Lipkis 81] Lipkis, T., *Consul Note 7: "Producing Explanation Responses"*, 1981.

[Mark 81] Mark, W., "Representation and inference in the Consul system " in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 1981.

[Mark 82] Mark, W., "Realization," in J. G. Schmolze and R. J. Brachman (eds.), *Proceedings of the 1981 KL-ONE Workshop*, pp. 76–87, May 1982.

[Sondheimer 84a] Sondheimer, N. K., R. M. Weischedel, and R. J. Bobrow, "Semantic interpretation using KL-ONE," in *Proceedings of Coling84*, pp. 101–107, Association for Computational Linguistics, July 1984.

[Sondheimer 84b] Sondheimer, N. K., *Consul Note 24: "Translating to User Model"*, USC/Information Sciences Institute, 1984.

[Wilczynski 81] Wilczynski, D., "Knowledge acquisition in the Consul system," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 1981.

END

FILMED

6-85

DTIC