# Knowledge Acquisition in The Consul System[1]

David Wilczynski

USC/Information Sciences Institute
4676 Admiralty Way
Marina Del Rey, California 90291

Abstract

Many knowledge-based systems feature general machinery that operates on externally supplied information, These systems must solve the acquisition problem: how to represent the external knowledge, determine if it is adequate, and incorporate it into the knowledge base. As a mediator between users and interactive services, the Consul system must understand the intent and behavior of programs that perform interactive functions To Consul, understanding a function means classifying a description of it in a highly structured, prebuilt knowledge base. A special formalism has been designed in which a service builder both programs functions and describes their actions. The resulting functional descriptions are then translated and interactively classified into Consul's knowledge base by Consul s acquisition component The acquisition dialogue with the service builder will be shown to be robust with respect to the information provided by the service builder. Inference rules are automatically generated to account for discrepancies between a program's specifications and expectations derived from Consuls knowledge base.[2]

## 1. Introduction

The Consul system ([3], [6] in this proceedings) is being designed to support natural, helpful, and consistent interactions between users and online services. In order to provide these cooperative facilities, any user interface must have a great deal of knowledge about the functionality of its services. However, unlike attempts at enhancing *specific* tools, Consul is being constructed as a general framework in which a wide variety of services can be embedded. The necessary functional knowledge is carefully organized within a central knowledge base comprising (1) *user* knowledge, service-independent, user independent knowledge relating basic user needs to user actions, objects, and English language expressions, (2) *systems* knowledge, a service-independent representation of the detailed behavior of the basic operations found in any service (e.g.. moving, deleting, scheduling), and (3) *service* knowledge, an instantiation of the systems knowledge to the actual operations and data structures of some interactive service implemented in Consul.

This research is supported by the Defense Advanced Research Protects Agency under Contract No DAHC15 72 C 0308. ARPA Order No 2223 Views and conclusions contained in this papet are the author s 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

[2]Many of the ideas in this paper were developed m collaboration with Robert Lingard

Except for service knowledge, all concepts and relations in the knowledge base are built in. Because of the complications of formally integrating service-dependent information, Consul has a component to *acquire* this information from the service builder. Consul relies on proper classification throughout its knowledge base, the relations between the service-independent concepts are carefully built in; the proper relation between the service and systems knowledge is the responsibility of the acquisition component. It is only through proper classification that a function's intent and behavior can be derived; having that information is the basis of Consuls cooperative features. Many of Consul's interactions with users rely on application of service-independent inference rules to map between descriptions in the knowledge base [5]. If the service-dependent information can be organized within the system model framework, those inference rules will automatically apply. The interface consistency that Consul provides is also a direct result of acquiring service-specific knowledge in terms of the carefully worked out systems model.

## 2. The Acquisition Process

A service in Consul is defined solely by its data structures and process scripts. Process scripts are programs that consist of two parts: a procedure body to perform some action and some descriptive information about that procedure, its parameters, its output, etc. The simplest process scripts, called process atoms, have a procedure body which consists of only a single call to some application code that is not further described to Consul.

Process scripts are the basis of a software design methodology for buiiding interactive tools [4]. Using this method, a tool builder explicitly differentiates between code that is "interesting" to the end user (implemented using the process script language) and code that is not (hidden as the bodies of process atoms). Informally, the code implemented via process scripts is the highest level code generated using a top-down or composite system design methodology. The information about the programs of the service being implemented comes solely from the process scripts and the acquisition dialogue with the service builder; it is not deduced by analyzing the code called by process atoms.

The acquisition of a service's process scripts occurs in four steps:

1. translating the process script into a knowledge base structure;

2. building a potential description of that structure from an initial classification by the service builder;

3. refining the potential description into a validated description. ie.. one that is consistent with the knowledge base:

4. building inference rules that relate the actual process script parameters to their roles in the validated description.

The goal of the acquisition process is recognition of the input structure in terms of the existing model, or in our terminology. *classifying* it in the Knowledge base. Step 3 above requires interaction with the service builder in order to to extract information about the script that is not represented explicitly

Step 4 is necessary because functions commonly name parameters that are not actually manipulated by the function. Instead, due to a desire for brevity, simplicity, or information hiding, these parameters act as access paths to the actual data structures to be processed. For example, a person may talk about "dialing Joe on the phone," instead of the more precise "dialing Joe's *phone number* on the phone." The casual, simpler statement is understandable if a unique phone number is associated with a person and if that association is known to be applicable in the context of dialing phones. Since the validated description shows how the process script parameters are actually used, the necessary inferential knowledge to account for this metonymic style can be explicitly represented.

## 3. Acquisition Requirements

The Consul approach is predicated on a detailed, coherent knowledge base. Given the complexity of the knowledge base, a service builder would find it impossible to update it correctly when adding new process scripts to the system. Consul therefore provides a semiautomatic acquisition process to aid the service builder. Major requirements of this process are:

It supports different implementations. Consuls model of interactive functions does not impose any implementation decisions on the service builder. On the contrary, the knowledge base will be shown to be 1) sufficiently general to support services whose functions depend on fundamentally different architectures and 2) detailed enough to support cooperative interactions with the end user.

The service builder need not know all the details of the knowledge base. The service builder uses the process script language to write his programs and an object definition language (still to be designed) to describe his objects. The acquisition process has the responsibility of translating his input into the system's representation, mapping it into Consul's knowledge base, and accounting for any inferential relationships discovered during acquisition.

Acquisition must report anomalous conditions to the service builder in a way that will elicit an appropriate response. A number of anomalous situations can arise during acquisition: inconsistency in an object's definition may not be detectable until that object is used in a process script; a new process script may conflict with, specialize, or generalize existing scripts; a process script may be imprecise in its input specifications (as is the case in the scenario to be presented). Whatever the situation, the acquisition process must give the service builder the information that not only explicates the problem, but induces a helpful response.

Consul's acquisition process is intended to be robust, i.e., to successfully model service-specific information without overly constraining the service builder. The next section shows'that this goal is achievable.

## 4. An Acquisition Scenario

The service-dependent concepts in this scenario are modelled after those in the SIGMA message system [8]. Figure 5-1 shows how the SIGMA concepts, prefaced by Sg, relate to the service independent concepts in the knowledge base[3]

In SIGMA, messages are kept in a central data base. A user is never sent messages. Instead, he gains access to them through citations delivered to his pending file, his personal mailbox. Besides having a pointer to the referenced message, the citation also contains addressing information, the subject of the message, and other summary information. Because a message can be "sent" or "forwarded" for different reasons (for review, for action, for information, etc.), the citation type is used to identify the nature of the transmission.

The SIGMAForward process script to be acquired (Figure 5-2) sends a citation of the SgOpenMessage -the message currently being worked on by the SgloggedOnUser-to a designated user The input to the script is the SgCitationType and the name of the intended recipient, SgUser. Since this scenario will be concerned only with the Input and DataStructuresAccessed descriptors of the process script, only they appear as roles in the translation shown in Figure 5-3. Notice that the process script itself has no statement of intent, since intent is derived solely through proper classification within the knowledge base

Figure 5-4 shows the service-independent part of the knowledge base relevant to the acquisition of SIGMAForward It defines a *Send Operation* as a specialization of *Move:* there is a

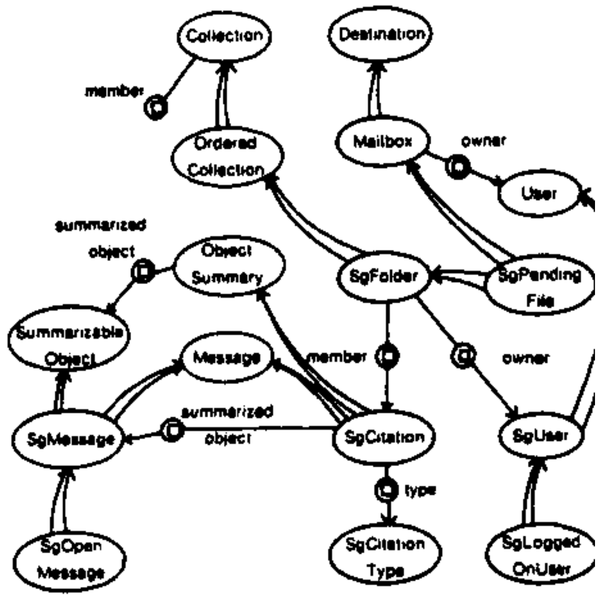


**Figure 5-1: Object Definitions**

---

Consuls knowledge base is implemented in the KL ONE formalism [1]

```
ProcessScript   SIGMAForward;
 Input                     u:SgUser, ct:SgCitationType;
 Output                    none;
 DataStructuresAccessed    SgOpenMessage,
                           SgLoggedOnUser;
 Preconditions             SgOpenMessageSV = true;
 SideEffects               none;
 Undo                      none;
 Error Conditions          e:NoMailBoxForRecipient;
  Call SIGMAForwardCode(u,ct);
```

**Figure 5-2:** SIGMAForward Process Script

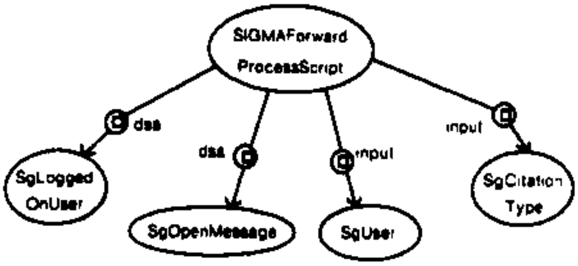


**Figure 5-3:** The Process Script Translation

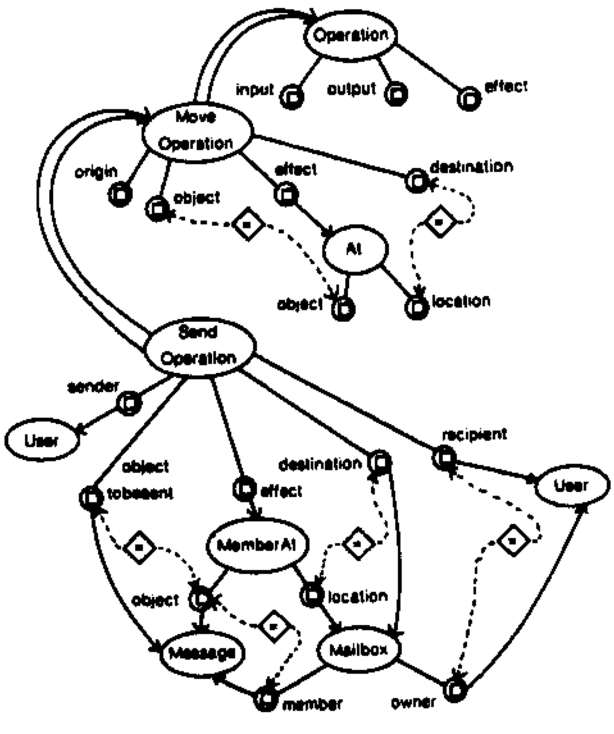


**Figure 5-4:** The Model of the Move and Send Operations

*sender,* the *object to be sent* is a *Message',* the *destination* is a *Mailbox* whose *owner* is the *recipient* of the *Message,* the effecf of the Send is that the object is *At* a *location* which is the *destination* of the Send and that the object being sent is a valid *member* of the Mailbox.

The following scenario shows the process of building a description of the SIGMAForward script in terms of the modelled Send operation. It results in the informal notion of "sending messages to people" being redescribable as an invocation of the SIGMAForward function.

### 4.1. The Acquisition Dialogue

The scenario starts as the translation of the process script (Figure 5-3) is presented to the acquirer.

Acquirer: WHAT KIND OF AN OPERATION IS THIS? (gives the service builder a list of top level system operations to choose from)

Service Builder: Move

Acquirer: WHAT KIND OF A *Move* IS THIS? (gives the service builder a list of Kinds of *Move* to choose from)

Service Builder: Send

The acquirer builds a potential *Send* operation *SIGMAForwardOperation* (Figure 5-5) with the SIGMAForward process script as the filler of the *procedure* role. It then tries to match the roles of the process script with those of the operation it has just built.

Acquirer: WHAT IS THE *destination*?*

Service Builder: u:Sg(Jser

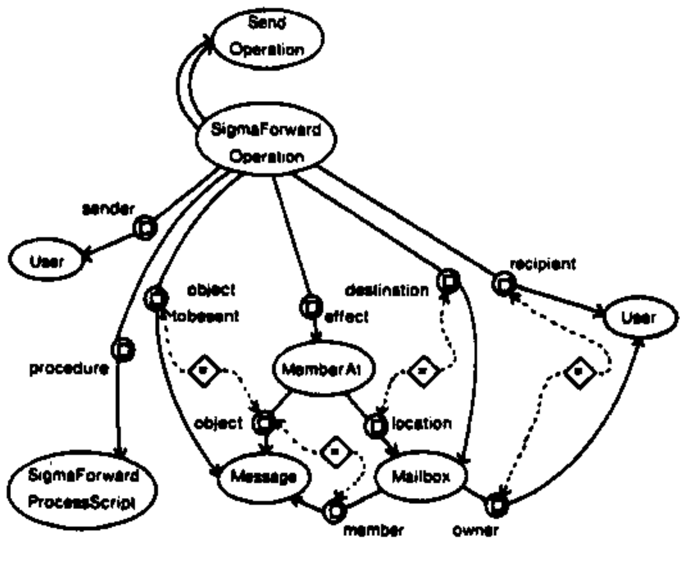


**Figure 5-5:** SIGMAForward as a Potential Send Operation

Acquirer: (examines knowledge base (Figure 5-1) to see if a *uSgUser* is a kind of *Mailbox)*

THE *destination* OF A *Send* HAS BEEN DEFINED TO BE A *Mailbox. u:SgUser* IS NOT KNOWN TO BE A *Mailbox.* IS IT?

Service Builder: No

Acquirer: CAN YOU GIVE ME MORE INFORMATION ABOUT THE *destination* of *Send?*

Service Builder: The destination of Send is the SgPendmgFile whose owner is the u:SgUser.[4]

Acquirer: (examines knowledge base to see if So.Pending.File is a kind ofMa/7oox; since it is, the destination role is validated)

If SgPendmgFile were not classified as a Mailbox, we could add that relationship as part of this scenario (being aware that the addition of a new relation may have effects beyond the current acquisition session). We expect that many service-dependent relations will be discovered through their use and hence built during process script acquisition.

Acquirer: (because the "recipient" and the "owner" of the Mailbox are coreferential[5], the uSgUser is expected to be the recipient)

I ASSUME THAT *u:SgUser* IS THE RECIPIENT. IS IT?[6]

Service Builder: Yes.

Acquirer: WHAT IS THE *sender?*

Service Builder: SgLoggedOn User

Acquirer: *(sender* role validated since SgLoggedOnUser is a kind of User)

WHAT IS THE *objecttobesentf*

Service Builder: SgOpenMessage

Though the SgOpenMessage is a satisfactory filler for the objecttobesent role, its coreferentiality to the object of the Member At causes a problem.

Acquirer: THE *effect* OF YOUR PROCESS SCRIPT IS THAT THE *SgOperUessage* WILL BE *At* THE *SgPendmgFile* OF THE *u SgUser.* I EXPECT THE *SgOpenMessage* TO BE A VALID MEMBER OF THE *SgPendmgFile.*

In order for the Member At effect to be satisfied, the *location* must be a Mailbox whose *member* is the *object* of the MemberAt, The problem here is that SgPendingFiles contain SgCitations as members, not SgOpenMessages; so, sending an SgOpenMessage to it is *invalid* Several things could be wrong: SgOpenMessages could be valid members (the service builder simply hasn't told us yet); the SgPendingFile is not the destination of the Send: the SgOpenMessage is not the object of the Send (the actual case here). The acquirer must frame these possibilities in a question or questions that will elicit the right response.

Acquirer: IS THE *SgOpenMessage* A VALID *member* OF *SgPendingFile?*

Service Builder: No

Acquirer: CAN YOU GIVE ME MORE INFORMATION ABOUT THE *objecttobesent?*

Service Builder; The objecttobesent of the Send is a SgCitation whose summarizedobject is the SgOpen Message.

Acquirer: (revalidates the *objecttobesent* role; validates the *effect* role; finds that one of the process script parameters is still unaccounted for)

WHAT IS *ct.SgCitattonType* USED FOR?

Service Builder: The ct.SgCitationType is the type of the SgCitation.

Acquirer: (builds the correspondence between *ct.SgCitattonType* and the the *type* role of the *SgCitation)*

SIGMAForward VALIDATED!

At this point the description shown in Figure 5-6 is complete and can be classified in the knowledge base The acquisition process has successfully discovered the intent of the process script via the above dialogue. Now it remains to account lor the relationship between the the process script's parameters and roles of the validated description.

## 4.2. Building the Inference Rules

The validated description of the SIGMAForward process script shows that there is an implied access path from a SIGMA user to his pending file, and one from an open message to its citation. Because of these relationships, the following statements are logically equivalent (assume that "Forward" implies a certain citation type):

1. Forward the open message to Jones.

2. Forward the open message to Jones' pending file.

The Jargon parser [9] accepts input like this and produces forms from which we generate the appropriate KI-ONE We are assuming this class of input to be sufficient for now.

Shown in Figure 5-5 by the links to the <£> structural description

The Teiresiaa extension (2) to the Mycin (7) rule-based system generates expectations during rule acquisition by comparing the new rule being acquired to existing ones. Aa pointed out by Davis, because Teiresias has no domain model, some of those expectations *ara* wrong because coincidental correlations could not be separated from reeJ ones In Consul, all expectations are based on characteristics of the built-in systems model

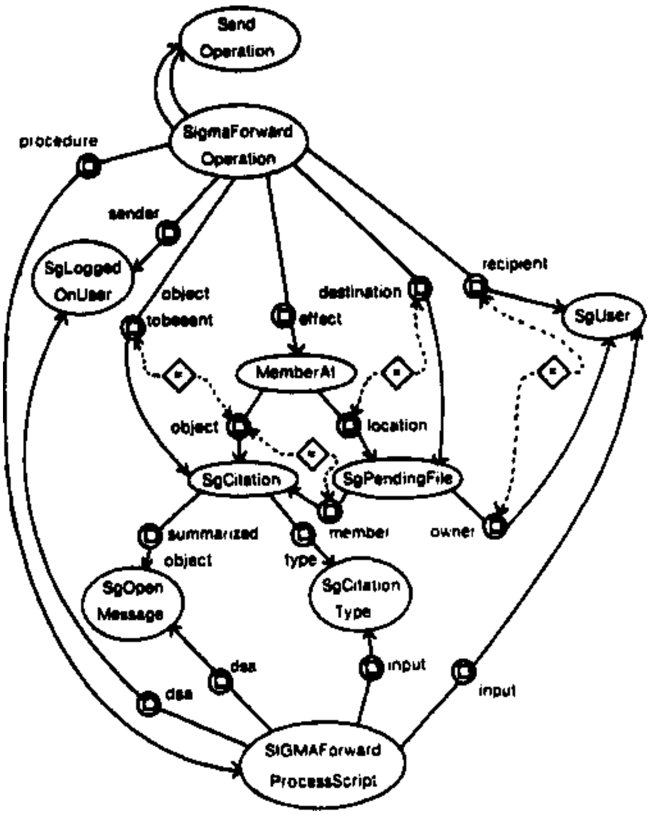


Figure 5-6: The Validated Description

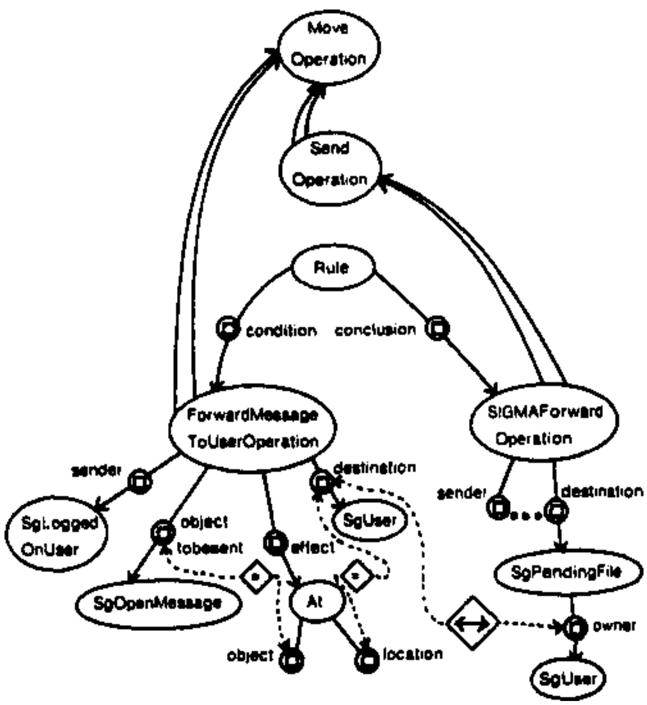


Figure 5-7: An Inference Rule Built at Acquisition Time

3. Forward a citation of the open message to Jones.

4. Forward a citation of the open message to Jones' pending file.

The last statement would instantiate the validated description of Figure 5-6 and is therefore executable. The first (and briefest one) matches the specification required by the process script, but not that of the validated description. Since statement 1 should cause the same process script to be invoked, inferential knowledge is needed to map it into the validated description. The necessary inference rule, shown in Figure 5-7 shows how a description of a Move operation whose object to be sent is a SgOpenMessage and whose destination is a SgUser can be transformed into the validated description[7]. Similar rules are constructed to account for statements 2 and 3.

The inference rules are completely determined by the relation between the parameters as defined in the process script and the roles of the operation as defined by the service-independent model. Whenever the correspondence is not direct, inference rules need to be generated. Notice that in comparison to the validated description, the condition parts of these inference rules are always less precise, potentially less constrained and may (as in this example) have a more general classification. These factors combine to allow the end user a certain amount of informality in making his requests. However, they may also lead to problems. For example, the generation of a new rule may introduce ambiguities when distinct validated descriptions produce identical rule conditions. This and other issues, such as using rules to represent defaults, giving the service builder some control of the rule generation process, and recognizing and handling cases when new rules specialize or generalize existing ones, are currently under study.

## 5. Conclusions

The prototype acquisition component described in this paper just begins to address the requirements specified in section 3. We have, however, shown that acquisition has the capabilities to make the Consul approach feasible. The service builder can write his functions in a language suited to his task and have them interactively assimilated into Consul's knowledge base. Any inferential knowledge needed to account for service-dependent conventions is automatically generated.

What remains to be shown is the robustness of the process. Even though the scenario showed how the acquirer directs the dialogue, how it detects some anomalous situations, and how it can make certain assumptions, many issues remain. Robustness is a critical concern if the acquirer is to interact with real service builders and not Consul specialists.

The task of building a cooperative user interface is too large and complex to attempt for each new service. One of our goals is to build a framework in which many new services can be embedded. Acquisition, as described in this paper, is a fundamental part of that framework.

The ↔ structural description indicates how roles in the rule condition corresponds to roles in the rule conclusion In order to simply Figure 5 7. only one of them is shown.

# References


[1] Ronald Brach man.
*A Structural Paradigm for Representing Knowledge.*
Technical Report, Bolt, Beranek, and Newman, Inc., 1978

[2] Randall Davis.
Interactive Transfer of Expertise: Acquisition of New Inference Rules.
In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.* IXAI, 1977.

[3] David Wilczynski, William Mark, Robert Lingard, Tom Lipkis.
Cooperative Interactive Systems.
In 7980 *Annual Technical Report,*. USC/Information Sciences Institute, 1981.

[4] Robert Lingard.
A Software Methodology for Building Interactive Tools.
In *Proceedings of the Fifth international Conference on Software Engineering*. 1981.

[5] William Mark.
Rule-Based Inference In Large Knowledge Bases.
In *Proceedings of the National Conference on Artificial Intelligence,* American Association for Artificial Intelligence, August, 1980.

[6] William Mark.
Representation and Inference in the Consul System.
In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* IJCAI, 1981.

[7] Edward H. Shortliffe.
*MYCIN: A Rule Based Computer Program for Advising Physicians Regarding Anti-Microbial Therapy Selection.*
Technical Report AI Lab Memo AIM-250, Stanford University, October, 1974.

[8] ' R. Stotz, R. Tugender, D. Wilczynski, and D. Oestreicher.
SIGMA: An Interactive Message Service for the Military Message Experiment.
In *Proceedings of the National Computer Conference.* AFIPS, May, 1979.

(9] W. A. Woods.
*Theoretical Studies in Natural Language Understanding. Annual Report.*
Technical Report BBN Report No. 4332, Bolt, Beranek, and Newman, Inc., 1979.