# PHRAN–SPAN: A NATURAL LANGUAGE[1] Interface For System Specifications

**John J. Granacki**
Dept. of EE-Systems
Univ. of Southern Calif.
Los Angeles, CA 90089-0781

**Alice C. Parker**
Computer Science Dept.
Univ. of Southern Calif.
Los Angeles, CA 90089-0781

**Technical Report CRI-87-11**
February 2, 1987

[1]Accepted to the Proceedings of the 24th Design Automation Conference. Revised 12/26/86.

## 1. Abstract

This paper describes a natural language interface, PHRAN-SPAN, for specifying the abstract behavior of digital systems in restricted English text. A neutral formal representation for the behavior is described using the USC Design Data Structure. A small set of concepts that characterize digital system behavior are presented using this representation. Finally, an intermediate representation based on Conceptual Dependencies is presented. Its use with a semantic-based parser to translate from English to the formal representation is illustrated by a series of examples.

## 2. Introduction

This work describes the PHRAN-SPAN[1] natural language interface to ADAM, the USC Advanced Design AutoMation System [Granacki 85]. PHRAN-SPAN is used to capture system specifications, with particular emphasis on the abstract behavior of the system being specified.

### 2.1. The ADAM System

ADAM is an integrated computer aided design system intended to aid a designer by providing a unified framework which combines: a natural language interface for specification; a database for both the design representation and a collection of knowledge bases; and a knowledge based planner [Knapp 86], which supervises specialized procedures. The major subsystems of ADAM are a suite of custom layout tools, an expert system which aids in the design of testable circuits [Breuer 85], and a knowledge-based synthesis subsystem.

---

[1] PHRasal ANalysis-SPecification ANalysis

structured, more costly hardware design or they simply cannot be used to specify the behavior of complex systems composed of large numbers of components and many levels of hierarchies.

### 3. Relationship to Other Research

Previous work done on processing natural language specifications has been concerned primarily with software systems [Balzer 85, Mander 79], programs [Abbott 83, Ginsparg 77] and data types [Comer 79]. This work falls into two categories. The first is characterized by virtually unrestricted application domains and therefore requires enormous vocabularies and the ability to deal with tremendous variability in the input. The second covers a very limited domain; namely, the manipulation of the objects which were created from the specification, e.g. CREATE A STACK, DELETE A SET, etc. Also, it should be noted that the research described in the paper by Mander was only concerned with syntactic analysis.

To make the problem tractable, for this research we selected an intermediate path and chose a limited domain, the behavior of digital systems. In addition, this system expects a structured input that has been checked for spelling errors and possibly mistypings.

One prior endeavor involved the application of natural language processing as an input to a design system for digital electronics, [Grinberg 80], but this work actually focused on the construction of a circuit given predefined components and was focused on implementation rather than specification. Furthermore, it used certain hyphenated verb forms, e.g. IS-CAPTURED-IN, and noun phrases like NUMBER-OF-WORDS to aid in the processing making it more like an application-oriented programming language.

Other recent work, like the UNIX Consultant (UC) [Wilensky 84], and CLEOPATRA [Samad 85], answer questions concerning a given body of knowledge, the former the UNIX operating system, the latter the results of a digital simulation. The PHRAN interface incorporated in this interface was developed for UC.

4. a parsing technique to map the natural language into the formal behavioral representation.

### 5.1. The Corpus

The corpus for this natural language interface was developed by acquiring actual specifications, having students write specifications and constructing additional examples. These examples were based on a taxonomy of high-level system behavior and a 2000+ word lexicon which were developed as part of this research.

Examples of the sentences taken from the actual specifications are provided in the following list:

1. *A block of data bytes is transferred by a sequence of data cycles.*
2. *The peripheral equipment shall sample the EF code word which is on the OD lines.*
3. *Each requestor communicates with the arbiter via two lines, a request line and a grant line.*
4. *Select shall be dropped 100 ns after the write is begun.*

### 5.2. The Corpus' Knowledge and the Parsing Technique

The representation of the knowledge expressed in this corpus was constrained by the choice of a pre-existing semantic-based parsing technique which was implemented by Arens in PHRAN, a PHRasal ANalysis program, [Arens 86]. PHRAN is a knowledge-based approach to natural language processing. The knowledge is stored in the form of pattern-concept pairs. A pattern is a phrasal construct which can be a word, a literal string, **Digital Equipment Corporation**, a general phrase such as

<component><sends><data>to<component>

and can be based on parts of speech, for example, <noun-phrase><verb>.

Associated with each phrasal pattern is a concept. The pattern-concept pair (PCP) encodes the semantic knowledge of the language. For example, associated with the pattern:

<component><sends><data>to<component>

4. **Physical** (Pss), which covers the physical hierarchy of components and the physical properties of these components. In this subspace there are two primitive object types: blocks and nets.

### 5.3.1. TSss arc types

The four types of timing arcs are sigma ($\sigma$) arcs, theta arcs ($\theta$), chi ($\chi$) arcs, and delta ($\delta$) arcs. A **sigma** arc represents an interval of time (or range [Knapp 83]) in the TSss. A **theta arc** represents a temporal constraint. A **chi arc** represents a casual relationship. A **delta arc** represents *inertial delay* [Breuer 76].

### 5.3.2. TSss node types

There are seven types of nodes in the TSss: pi ($\pi$) nodes, beta ($\beta$) nodes, gamma ($\gamma$) nodes, mu ($\mu$) nodes, rho ($\rho$) nodes, ($\alpha$) nodes, and omega ($\omega$) nodes. The $\pi$ node is a simple node that may *join* two arcs. A **beta node** represents an *and fork* point [Conway63] or a *cobegin* [Dijkstra 68]. A **gamma node** represents an exclusive-or branch. A **mu node** represents an *and join* point. A **rho node** represents an *exclusive-or join* point. For the $\gamma$ and $\rho$ nodes, one branch (arc) will actually be active in a properly specified behavior. An **alpha node** represents the beginning of a repetitive interval or loop. The arc or sequence of arcs that emanates from this point will eventually terminate in an **omega node** that represents the *normal* termination of the repetitive interval. There is at least one **omega node** for each **alpha node.**

The two types of predicates attached to timing arcs are

- **synchronous** predicates, and
- **asynchronous** predicates.

A synchronous predicate is attached to each of the arcs emanating from a gamma node. This predicate indicates the branch to be chosen at the time of *execution.* This corresponds to the familiar *if-then* conditional statement of most programming languages.

An asynchronous predicate is used to indicate a branch from a particular timing subgraph at any point in that subgraph, whenever the predicates become true. **Reset,**

- Dual Temporal Relation (DTR).

Control

- If-then-else,
- While,
- Repeat, and
- Looping.

Declarations

- Assignment or Inheritance Statements, and
- Structural or Physical Interconnection.

Abstractions of DDS Relations

- Value-Carrier-Net-Range (VCNR), and
- Operation-Module-Block-Range (OMBR).

Formal semantic definitions of all these concepts have been developed using the DDS as a modelling tool [Granacki 86]. The Unidirectional Value Transfer (UVT), the Single Temporal Relation (STR) and the Asynchronous Temporal activity (ATA) are described below.

### 5.5. The Unidirectional Value Transfer

An example of these models is the DDS template for a UVT shown in Figure 5-1 and Table 5-1.

This template spans two of the DDS subspaces, the DFss and the TSss. The template for the UVT in the DFss is composed of three values and three operations and their data link arcs. The **control** operation may be associated with the **source** operation, where the data flow value, **info** originates or the **sink** operation, the destination for the data flow value, **info** or the **control** may be associated with a third independent operation.

An example of a sentence which maps into a UVT is

**The cpu transfers the block of data bytes from the disk to the control store.**

If no timing information or constraints are specified in the same sentence then the TSss template shown in Figure 5-1 is used as the default. The default TSss template shows the timing for the three operations and the necessary constraints for a valid UVT. For example, the constraints labelled $\theta_1$ and $\theta_2$ represent the fact that the time interval for the operation **control** must begin before the end of the intervals associated with the source operation and the sink operation. If these constraints were not present it would be meaningless to associate the **src_cntl** value or the **snk_ctl** value with this particular transfer of the value **info.**

The fact that these constraint arcs emanate from a node labelled $\beta_{c2}$ indicates that the two constraints, $\theta_1$ and $\theta_2$ and the interval labelled $\sigma_4$ all begin concurrently.

### 5.6. The Single Temporal Relation

Additional control or timing information can be added in the same sentence. For example, the sentence may be modified as follows:

**The cpu transfers the block of data bytes from the disk to the control store in less than 100 ns.**

The adverbial phrase, **in less than 100 ns** would result in an Single Temporal

The subject of the sentence must belong to the semantic category of an **a_component** (abstract component) *or* a **df_opn** (data flow operation) for this pattern to match. The next part of the pattern indicates that some verb form with the root of **transfer** must be present. The verb may be in a different tense, *e.g.*, transferred or it may be combined with a modal verb like **shall**. The object transferred belongs to the semantic category **df_val** (data flow value). The abstract component is introduced to handle a certain ambiguity that arises from specifying a component in English. For example, a **cpu** may be a logical module or a physical block in the DDS. An additional declaration or phrase like an appositive is required to resolve this type of ambiguity. Therefore, when the word **cpu** is encountered it is treated as an abstract component until additional information is provided. If a **cpu process** had been specified this would be interpreted more precisely as a **df_opn** and the pattern would also match.

Associated with each pattern is a concept that describes the meaning of the word phrase or sentence that matches the pattern. The concept is represented as a frame [Winston 84] using the specification representation language SRL, based on the set of concepts we introduced.

For example, the concept part of the pattern-concept pair for the UVT in SRL is

```
(uni_dir_vtrans
        (source  (a_component ?source))
        (sink    (a_component ?sink))
        (info    (df_val ?info))
        (control (a_component ?control)))
```

The facet slots in the frame are represented by variable names that are prefixed with a question mark. Fillers for these slots are obtained when the sentence matches the pattern. For example, consider the sentence,

**The cpu transfers the command.**

When the sentence is processed tokens are created for the cpu and the command. The resulting concept for this example is

### 5.9. Non-Noun Phrases and Ambiguity

Some extensions to PHRAN were necessary to handle non-noun phrases and ambiguity.

It is quite common for nouns to be used as modifiers or other nouns in specifications. Some examples are *bus request cycle, transfer block size, interrupt vector transfer phase*, and *arithmetic register reference instruction.*

These phrases are often created by the specifier to reference a particular entity, e.g., piece of hardware, an activity, or a range of time. Therefore, their meaning can usually be inferred from the last noun in the phrase. However, the process of forming these groups of nouns into a specific noun phrase is complicated by the fact that many of the words used in specifications are syntactically ambiguous, i.e., the word may be either a noun or a verb. Examples of these words are **interrupt, process, signal, start** and **transfer.**

PHRAN's inherent priority scheme was used to solve part of this problem, *i.e.* the pattern with the shortest length is tried first, all else being equal. This results in a word that can be used as either a noun or a verb being recognized first as a noun. Then PHRAN's access routine was modified to look up all possible meanings of a word; therefore, the only problem left to solve was when to use the word as a noun or as a verb.

#### 5.9.1. Rules for disambiguation

Some potential rules for resolving the word's use in a sentence are

1. check the agreement in number of the subject (potential noun phrase) with the verb/noun,

2. check whether the word preceding the verb/noun is an *active agent*, *i.e.* a possible subject of the sentence and/or

3. check whether the word following the verb/noun is a verb or a noun or another verb/noun.

Evaluation of several examples led to a simple heuristic based on rule #1 and rule #3.

the **data-transfer-register1** and its meaning would be an **a_component.**

## 6. Current Status

The system currently recognizes simple sentences associated with all the primitive concepts of our specification language, *e.g.*, UVT, BVT, CTI and DTR which are required to describe behavior in the domain of digital systems. At the time of this writing (November 1986), actual pattern concept pairs have been built for 25 basic verb patterns common to specifications and 100+ nouns. In addition, the system uses several hundred of PHRAN's patterns as supplied from Berkeley and has added some auxiliary verb constructs for verbs like **shall** to the pattern concept pairs and the ability to handle certain type of noun-noun phrases prevalent in specifications. Also the system has been extended to detect ambiguity that can arise from the use of nouns and verbs that have the same lexical stem, *e.g.*, transfer, interrupt, and signal.

The system is coded in Franz Lisp and is running in interpreted mode on a SUN/2 workstation under SUN's operating system, Version 1.4 (UNIX BSD 4.2). Typical sentences take approximately 15 to 35 cpu seconds to process. No attempt has been made to optimize the code, run compiled code or port the application to a Lisp processor. Any or all of these speed-ups should result in an interface which could operate in near real-time.

## 7. Conclusions

A small set of concepts for system-level specification have been identified and their semantics defined. The usefulness of the DDS as a neutral formal representation for capturing the specification of system level behavior has been demonstrated. These two conclusions, taken together, would allow us to construct an interface to ADAM for system specifications, independent of whether the interface is natural language or formal language.

The system also has demonstrated the application of PHRAN to another domain and for a different purpose than previous applications.

## References


[Abbott 83] Abbott, R.J.
Program Description by Informal English Descriptions.
*CACM* 26(31):882-894, November, 1983.

[Arens 86] Arens.
*CLUSTER: An Approach to Contextual Language Understanding.*
PhD thesis, University of California, Berkeley, 1986.

[Balzer 85] Balzer, R.
A 15 Year Perspective on Automatic Programming (Invited Paper).
*IEEE Transactions on Software Engineering* SE-11(11):1257-1268, November, 1985.

[Breuer 76] Breuer, M.A. and A.D. Friedman.
*Diagnosis & Reliable Design of Digital Systems.*
Computer Science Press, Inc., Rockville, Maryland, 1976.

[Breuer 85] Breuer, M. and X. Zhu.
A Knowledge Based System for Selecting a Test Methodology for a PLA.
In *22nd ACM/IEEE Design Automation Conference*, pages 259-265. ACM/IEEE, 1985.

[Comer 79] Comer, J.R.
*An Experimental Natural-Language Processor for Generating Data Type Specifications.*
PhD thesis, Texas A&M University, May, 1979.
Computing Science.

[Dijkstra 68] Dijkstra, E.W.
Cooperating sequential processes.
In Genuys, F. (editors), *Programming Languages*, pages 43-112. Academic Press, London, 1968.

[Ginsparg 77] Ginsparg, J.M.
*A Parser for English and Its Application in an Automatic Programming System.*
PhD thesis, Stanford University, June, 1977.

[Granacki 85] Granacki, J., D. Knapp and A. Parker.
The ADAM Design Automation System: Overview, Planner and Natural Language Interface.
In *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, pages 727-730. ACM/IEEE, June, 1985.