REPRESENTING TEMPORAL INFORMATION FOR DIGITAL SYSTEM SPECIFICATION

Technical Report CENG 89-32

JOHN J. GRANACKI
INFORMATION SCIENCE INSTITUTE

ALICE C. PARKER
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-07981

Representing Temporal Information for Digital System Specification

John J. Granacki
Information Sciences Institute

Alice C. Parker
Department of Electrical Engineering
University of Southern California
Los Angeles, CA, USA

November 30, 1989

1 Abstract

This paper describes a representation for temporal information useful for natural language understanding of system specifications. The representation is based on the design data structure (DDS). The DDS is widely used in the ADAM Advanced Design AutoMation system, and contains four models of digital designs, data flow, control and timing, structural and physical. The paper describes the control and timing model in detail.

The natural language interface operates by matching patterns in the input sentences and then relating the patterns to concepts in its knowledge base. A number of concepts for digital system specification have been created. Three of these are described in the paper, with emphasis on the universal value transfer (UVT). Asynchronous behavior and timing constraints are also discussed.

The natural language interface is operational and has been tested on a number of sentences.

2 Introduction

Digital system specifications are usually written in English or other natural language. Most existing formal techniques for describing or specifying digital systems are incomplete, thus overly restricting the resulting 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. Until a suitable formal notation for system specifications has been developed, the only manner in which this information can be processed by machine is via a natural language interface.

Understanding system specifications is especially challenging because a large portion of the information is temporal. The behavior of these systems can be specified by one or more processes (independently executing environments) which compete and/or communicate. A process can be started asynchronously (whenever specified conditions become true); execute indefinitely; start, suspend and terminate other processes asynchronously; exclude other processes from executing; communicate with other processes; and be asynchronously terminated or suspended itself when some specified conditions become true. The (clock) rates at which these processes run may be different from process to process, i.e., not a multiple of any common fundamental clock. Processes communicate via shared data, synchronize at critical points, or compete for shared resources.

This work describes a representation for temporal information like that described above. The representation was developed as part of a natural language interface for ADAM, the USC Advanced Design AutoMation System [1], [2]. The representation is used to process specifications that describe the behavior of digital systems in restricted English text.

The *behavior* of a system is composed of three parts:

1. the data flow of the system,

2. the timing and sequencing of the system, and

3. the interrelationships between the dataflow and the timing and sequencing information.

This representation allows the information to be easily separated for analysis or synthesis tasks. For example, the data flow information can be used to determine operator precedence and the bindings can be checked to verify that the timing and sequencing does not violate the inherent precedence relations.

The representation used to describe a digital system's timing and sequencing integrates together four types of information, all of which define partial orders. This allows the information to be integrated together into a formal graphical model. The different types of information that give rise to partial orderings and an example of the corresponding English text that produces each are:

1. relative temporal ordering, *e.g. event-a occurs before event-b*;

2. metric time, *e.g.*, *the data transfer occurs 10 seconds after event-a*;

3. causality, *e.g.*, *the cpu starts the memory data transfer cycle* and

4. asynchronous activity, *e.g.*, *when the reset button is pushed.*

The fundamental representation of behavior as data flow, sequencing and timing and their interrelationships is used to define higher-level concepts that characterize digital system behavior. These higher-level concepts are used by a semantic-based parser to understand system specifications.

The outline of the paper is as follows. First, we introduce the ADAM system and give a brief overview of how designs are represented. Following the introductory material, related research is discussed in Section 3. Then, in Section 4, we detail the temporal representation used in ADAM. Section 5 describes how the natural language interface uses this representation to understand system specifications. Section 6 presents current status and conclusions.

2.1 The ADAM System

The major subsystems of ADAM are a design-for-testability subsystem [3] and a digital synthesis subsystem [4]. The ADAM synthesis subsystem is intended to assist 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, [5], which supervises specialized synthesis procedures. Design information is represented using the Design Data Structure (DDS), ([6], [7]).

The goals of the synthesis subsystem are to produce correct implementations representing a range of tradeoffs, to allow varying degrees of user interaction, to allow design to proceed incrementally, starting from a partially-complete initial design, and to produce designs which meet several kinds of constraints, including timing requirements.

2.2 Overview of The Design Data Structure

The Design Data Structure (DDS) mentioned above is the underlying representation for the Specification Representation Language (SRL) used in the natural language interface.

The Design Data Structure is a unified representation for design information which contains data flow behavior, control and timing behavior, logical structure and physical structure. The DDS representation is interpretable for simulation purposes, the graphs can be easily traversed to detect errors analytically [8], it supports description of families of designs, it allows the design effort to be partitioned in a well-defined manner, it corresponds directly to formal models of the synthesis process [9], it supports easy tool interfacing, and supports incremental design, user interaction, and backtracking [10]. In addition, it provides the necessary semantics to represent synchronous and asynchronous events, timeouts, delays, pipelining, and process priorities.

The Design Data Structure represents specifications and implementations using separate *models* for the dataflow behavior, timing and control behavior, logical structure, and physical structure. This allows both control and timing and data flow behavior to be modeled accurately. Each model may be hierarchically decomposed, but the hierarchies may not be isomorphic. By separating different aspects of design information into models, instead of lumping behavior, structure and timing into a single representation at each 'level' of design, consistency checking is simpler, and redundancy is reduced.

The four models can be described briefly:

1. **Data Flow**: represents data dependencies and functional definitions. It is a bipartite acyclic graph where one type of node represents the operations and the other type of node represents the values. The data link arcs which connect these nodes indicate the sources and sinks of the values. These graphs are equivalent to a single assignment programming language.

2. **Control and Timing**: represents timing, sequence of events and conditional branching. It is represented by a directed acyclic graph, which consists of nodes corresponding to events, and arcs which represent intervals and connect these nodes. To capture as much semantic information about the design as possible, four types of arcs and seven types of nodes are used to model various aspects of timing and control (for example, concurrency, choice and constraints).

3. **Structural**: represents the logical decomposition of a circuit. This subspace is similar to a schematic or block diagram. It consist of modules which are interconnected by carriers.

4. **Physical**: represents the physical hierarchy of components and the physical properties of these components. In this subspace there are two primitive object types: blocks and nets.

Items in one model are related to those of other models by bindings. For example, a binding can occur between the addition in the dataflow, the ALU in the structural model which is used to implement it, and the time range during which the addition is performed. All relationships between the models are explicitly delineated in this way. These bindings correspond to the 0-1 variables in a formal model of the synthesis process, supporting verification as well as synthesis [9]. The bindings also support cleaner partitioning of the design effort, since correspondences between design models are explicit. Incremental design is possible with the simple addition of bindings, and backtracking is supported by deletion of bindings. Users can interact by viewing, creating, or deleting bindings.

In the DDS, abstract behavior is represented using the data flow and timing models. Synthesis systems manipulate this information and also place additional implementation information in the structural and physical models to create the design. An advantage of the DDS is that both data flow and timing graph representations are employed, so that both aspects of behavior can be accurately specified, and hierarchical decomposition of both is possible. This is desirable because the design of data paths and control is commonly done separately to reduce complexity, and yet tradeoffs between the two are possible and often performed in human designs. Separating data flow from control and timing also makes the specification of pipelined designs easier.

One could produce a single combined graph from the data flow and control and timing models by using the bindings to 'snap together' the two graphs at some level of the hierarchy. (It should be noted that the two graphs at other levels of the hierarchy might not compose cleanly into a single graph due to the non-isomorphism of the hierarchies.) Thus, using two separates graphs supports a superset of the information that could be contained in a single graph. The bindings which maintain the correspondence between the two graphs are explicit, facilitating searching for concurrency and other design features.

In Section 4 of this paper, we will give a more formal description of the DDS control and timing model and then present some examples of temporal information with the corresponding DDS representations. These examples will highlight the ability to express a number of different aspects of system behavior, grouped into high-level concepts.

2.3 High Level Concepts in The Digital System Specification Domain

After studying many natural language specifications, a small set of concepts that characterize system-level behavior, constraints and other ancillary data were developed. These concepts provided an understanding of the representational requirements for the natural language interface and formed the basis for the interface itself. The concepts can be grouped into classes as follows:

Information Transfer

- Unidirectional Value Transfer,
- Bidirectional Value Transfer, and
- Nondirectional Value Transfer.

Temporal Activities

- Asynchronous Temporal Activity,
- Causal Temporal Initiation,
- Causal Temporal Termination, and
- Single Temporal Event.

Temporal Constraints

- Single Temporal Relation, and
- Dual Temporal Relation.

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 [7].

3 Relationship to Other Research

This research is related to other work done in the broad areas of temporal logic, temporal representations and natural language understanding.

3.1 Temporal Logic

Temporal logic [11] is an extension of standard logic to time-related propositions. Temporal logic has been used by several researchers [12], [13], [14], [15], [16], [17] to reason about temporal issues associated with programs, network protocols, and most recently hardware. The general level of abstraction presented by Bochmann, Moszkowski and Fujita is at a level where the individual states of the device must be identified and much of the reasoning is done in terms of signals and their levels. Also the description of the behavior mixes structural, data flow and sequencing and timing information in a way which makes it difficult to reason about any one of these individually. Other work by Schwartz *et al.* [18] which could be applied to this research has introduced intervals and used temporal logic to reason about these intervals.

Shoham [19] has introduced a new nonmonotonic logic, *chronological ignorance* to deal with causality. His problem is much broader in scope than our specification problem and his notion of *extended prediction problem* has been challenged as nonexistent by Rayner [20].

3.2 Representation of Temporal Information

In addition, to the classic view of temporal logic described in the previous section, other work on representing temporal information has been done in the area of artificial intelligence [21], [22], and the survey of research on temporal modelling by [23]. The work of Allen [24], [25], [26] is closest to the research done in extending the DDS timing and sequencing representation. The basic similarities are the notion of the temporal interval as a primitive and the characterization of the relationships between temporal intervals in a hierarchical manner using constraint propagation techniques. There have been several other papers written on the computational aspects of Allen's Temporal Representation [27], [28], [29] and [30]. Our research differs in that we have added a different notion of points and have extended the semantics of the relationships to reflect causality.

Sathia *et al.* developed an integrated representation of time, causality, activity, authority, constraint representation and ownership for the scheduling and planning domain. Their model is an activity/state network which includes much of the same temporal information found in our model and treats causality in a similar way. In contrast, we are not producing a state-based representation and our notion of unifying the temporal, data flow, structural and physical models through bindings is better suited to the specification problem.

3.3 Related Research on Natural Language Understanding

Previous work done on processing natural language specifications has been concerned primarily with software systems [31], [32], programs [33], [34], and data types [35]. This work falls into two categories. The first is characterized by virtually unrestricted application domains and therefore required enormous vocabularies and the ability to deal with tremendous variability in the input. The second covered 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 mistypings.

One prior endeavor involved the application of natural language processing as an input to a design system for digital electronics, [36], 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), [37], and CLEOPATRA, [38], answer questions concerning a given body of knowledge, the former the UNIX operating system, the latter the results of a digital simulation.

4 The DDS Control and Timing Model

The DDS control and timing model is used to describe the timing and ordering of events in hardware. It is formally represented by a directed acyclic graph (DAG) model. Informally, such a model is constructed of *ranges* (arcs) and *points* (nodes). There are four types of arcs in this model. The four types are based on the semantic use of the arcs in representing timing and sequencing information. There are also seven types of nodes in this model. Informally, a range represents a constraining or derived duration of time, an ordering relationship, or a causal relationship between points. Points represent events of infinitesimal duration, which separate ranges. A range may have either a known, an approximately known, or a wholly unknown duration; any two points may or may not be linked by a range. Ranges are directed: i.e. they indicate the direction of the flow of time. A nonhierarchical graph of points and ranges is a directed acyclic graph, corresponding to a partial ordering of events (hierarchical graphs appear as hypergraphs). The graph can convey loops, conditional branching and concurrent execution, all of which are important in hardware design.

4.1 DDS control and timing graph arc types

The four types of arcs are interval arcs, constraint arcs, causal arcs, and delay arcs.

An **interval arc** represents an interval of time (or range [39]). An interval arc may also be viewed as a sequence of events or points. However, since a point has no actual dimension (like a geometrical point on a line), the points serve only as labels used for reference to specific events. The duration or length of the interval is associated with the arcs joining the nodes. Since the ultimate objective is a physically realizable implementation, one cannot bind an operation or a value to a node or point; bindings are only permitted to arcs. Finally, interval arcs may be assigned a specific length that indicates a particular amount of time (units are established as required by the design) as shown in Figure 1. The ends of the interval in this figure are referenced as p_1 and p_2. [1] Note, this length is defined in terms of a value and a relation. The relation may be any of the following: $>$, $<$, $=$, $\geq$, $\leq$. There is no restriction that the length of an arc be positive; however, time proceeds in only one direction and negative lengths can be transformed to positive lengths by reversing the direction of the arc.

[1] In this representation a **p point** is a simple primitive event and will be explained in Section 4.2.

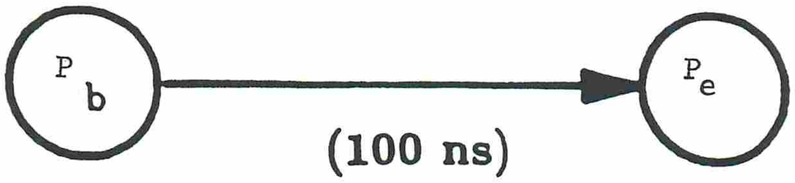

Figure 1: An interval arc in the DDS and its length in nanoseconds.

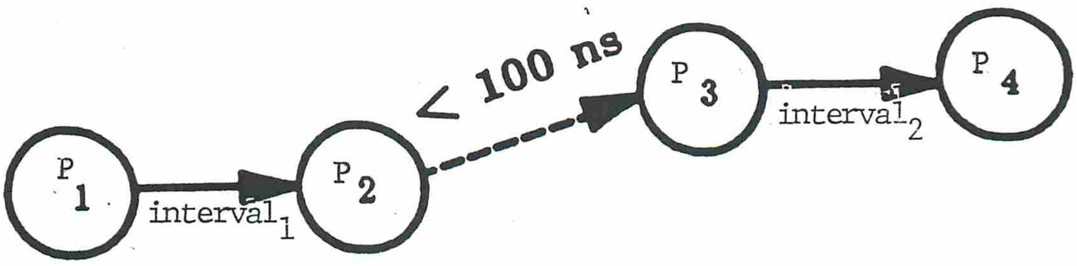

Figure 2: A constraint arc used to specify that one interval begins 100 ns after the end of the other.

A constraint arc represents a temporal constraint. For example, if the beginning of one interval is specified to occur at 100ns after another interval ends, a constraint arc is used to represent this information. An example of this is shown in Figure 2.

A causal arc represents a causal relationship. An example of this type of arc is where the end of $interval_1$ is causally related to the beginning of $interval_2$, *i.e.*, $interval_1$ ending causes $interval_2$ to start (shown in Figure 3).

A delay arc represents *inertial delay* [40]. [2] In the research presented here, a simplified model is used that lumps the various delays associated with a physical component. The lumped delay, $delay_1$, as shown in Fig 4 is associated with $delay_{in}$, that begins with the arrival of the last input and ends with the beginning of the output being valid, $delay_{out}$. The end of the interval, $delay_{in}$ is constrained by the arc labeled, $constraint > 0$ to precede the beginning of the interval labeled $delay_1$. This constraint simply stated is that the input value must not end before the operation begins. The analogy in a physical implementation would be to measure the output after the signal was removed

[2]Inertial delay is not equal to propagation delay.

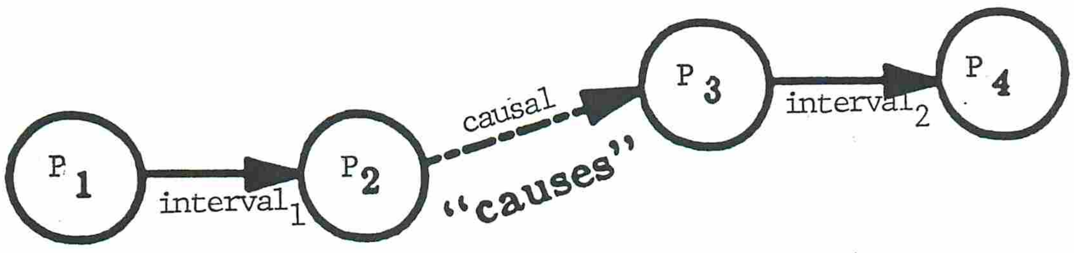

Figure 3: A causal arc representing a causal relationship.

and the input was floating.

Note that the DDS can be used to construct a detailed timing model at the transistor level, if required. Obviously, such a model is not required for system specifications. Semantically, a clock interval is not fundamentally different from an arbitrary regular repetition of **interval** arcs. The symbol **tau** (τ) will be used to refer to the fundamental unit of time chosen as a quantum unit for a particular interpretation. No interval of smaller duration than this may be distinguished under that interpretation.

4.2 Control and Timing Graph Node Types

Points are primitives and have no explicit definition within the model. There are seven types of points in the control and timing model: **p** nodes, **and** nodes, **or** nodes, **and-j** nodes, **or-j** nodes, **alpha** (α) nodes, and **omega** (ω) nodes. The first type is a simple node that may *join* two arcs, providing a label for the *meets* [3] relationship [24] or providing a label for an event. This is a **p** node or *point*. An example is shown in Figure 5. The location of p_i is within the interval between p_b and p_e but is not further specified.

The remaining six types of nodes are only useful to establish the temporal relationship between three or more arcs. The types of nodes will be described with respect to interval arcs only. The various combinations of nodes and arcs are defined in [7].

An **and node** represents a point at which the end of one interval is synchronously associated with the beginning of two or more other intervals. This may also be referred to as an *and fork* point [41] or a *cobegin* [42]. The two branches *begin* together but no additional information is implied in Figure 6 (Note: Allen [24] uses the label *starts*,

[3]Meets is one of the thirteen unambiguous relationships that Allen defines between any two intervals in time. It is a graphical relation in which the end of one interval abuts the beginning of the following interval.

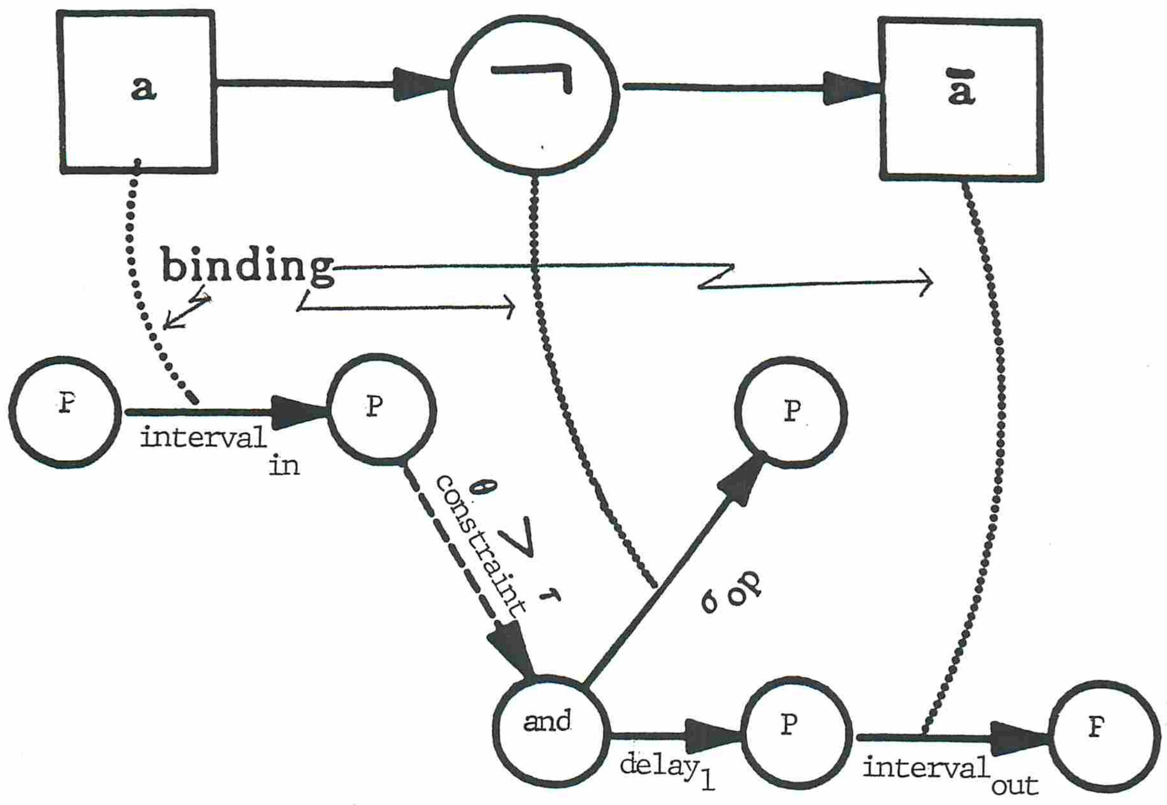

Figure 4: A DDS representation showing the use of a delay arc

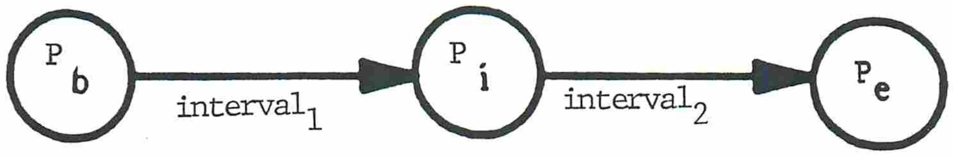

Figure 5: Three p points joined by two interval arcs.

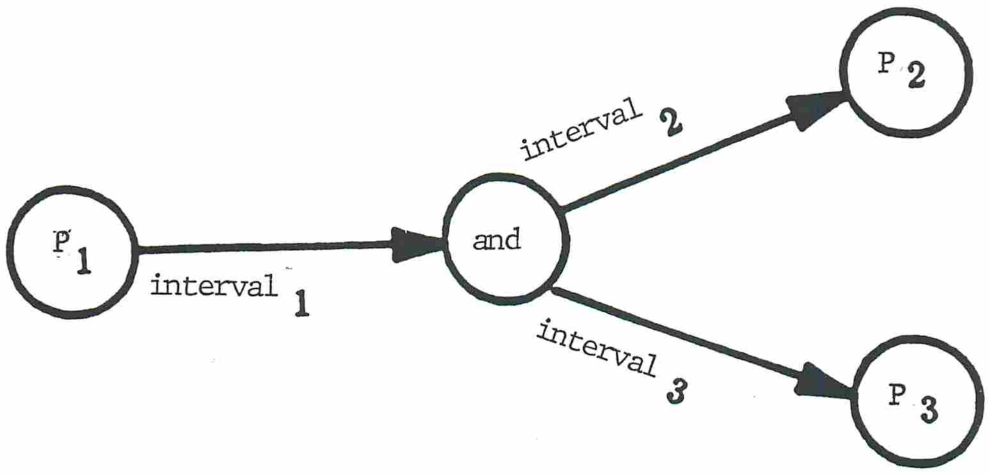

Figure 6: A two branch **and** fork example.

which seems to imply some causality; the model described here differentiates the causal information by using the causal arc construct.)

An or node represents a point at which the end of one interval is associated with one of a set of subsequent intervals, thereby representing an n-way branch. Each branch exiting from this node is an exclusive selection. The choice of branch is based on the value of a predicate that is attached to each arc emanating from the *or* node. The predicates will be discussed further in the next section with respect to their use in describing asynchrony and in the section on DDS canonical templates. An *or* node, two branch example is depicted in Figure 7.

An **and-j node** represents an *and join* point, *i.e.*, the termination of two parallel branches. This node is analogous to a *coend* [42]; appropriate delay is inserted in either branch to insure concurrent termination. An example of a **coend** is depicted in Figure 8. An **or-j node** represents an *exclusive-or join* point. The arcs that terminate at this point represent all possible branches that could be the predecessor of the arc emanating from the join. Only one branch (arc) will actually be active in a properly specified behavior. An example of an *or-j* node joining three interval arcs is shown in Figure 9.

Alpha nodes and **omega nodes** occur in pairs and will be described together. 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. The basic concept is shown in Figure 10 where the details of the control and timing graph loop body

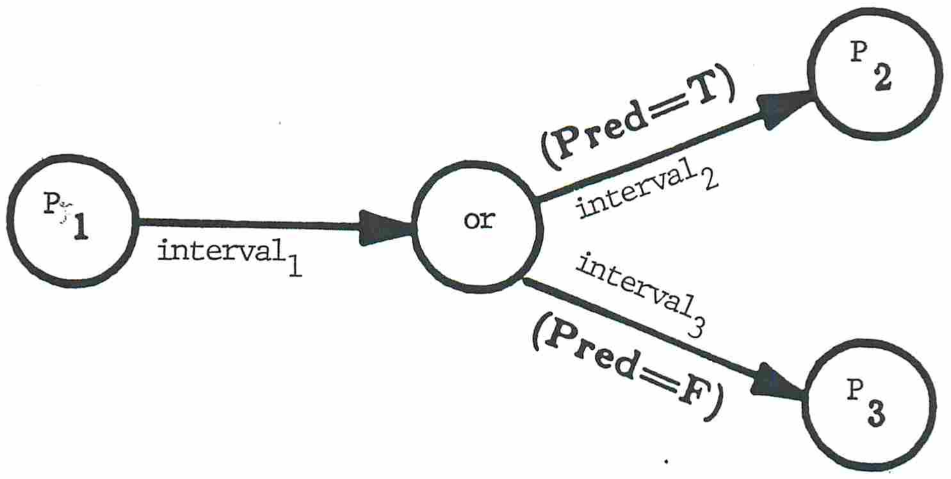

Figure 7: A two branch **or fork** example.

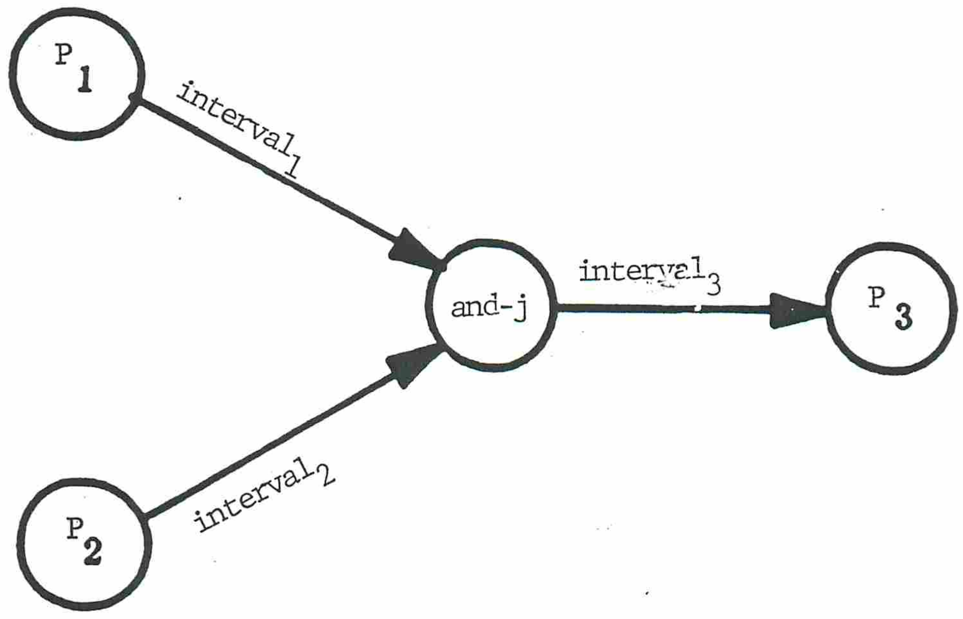

Figure 8: An example of **coend** in the Control and Timing Graph.

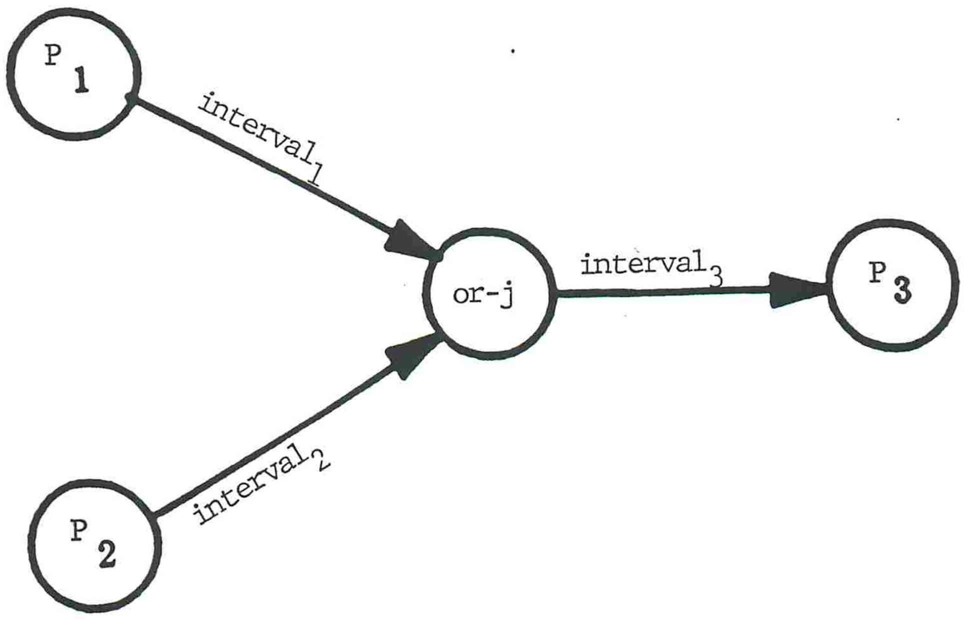

Figure 9: An example of **xor join** in the Control and Timing Graph.

are shown schematically. The *alpha* node and *omega* node are given symbolic subscripts. These subscripts are used in distinguishing values and operations in different iterations of the loop. When values are bound to a loop in the control and timing model, a correspondence between the value subscripts that are in parentheses () and the subscript of the loop is established. In effect, this loop could be considered to be *unrolled* in the DDS and is simply a sequence of subgraphs delimited by subscripted alpha and omega nodes as indicated in Figure 11. However, unfixed loops, *i.e.*, those loops with an unknown number of repetitions and infinite loops cannot be unrolled. Also, since the arcs inside

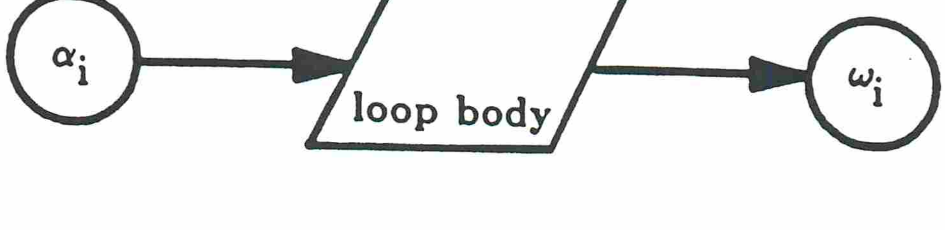

Figure 10: An iterative loop in the Control and Timing Graph.

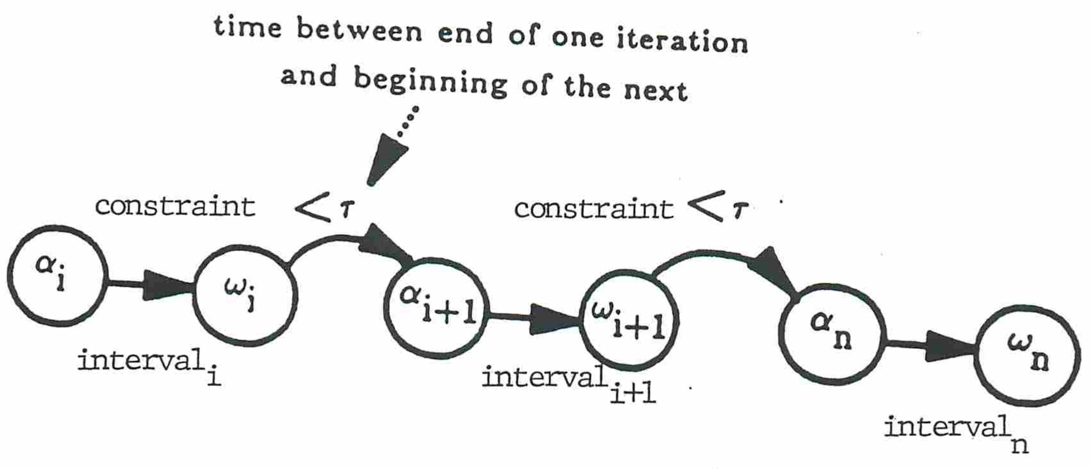

Figure 11: An iterative loop *unrolled.*

the loop are subscripted there may be a different length of time associated with every execution of the loop.

5 Representing Temporal Information in The Natural Language Interface

The use of natural language allows the ADAM system to capture a specification of the system's behavior from the designer's point of view with a minimum of implementation detail. In addition, the use of a natural language interface will relieve potential users of the burden of learning one or more artificial languages. This interface will also facilitate the construction of a complete, correct, consistent, concise and comprehensible specification, via an interactive dialogue with the designer. The interface will allow the user to interactively assist in the disambiguation of the the input, simplifying a difficult part of natural language processing.

The research described here differs from UC and CLEOPATRA in that it is creating a design entity, i.e., a formal, neutral representation of the behavior being specified. To create this representation, semantic knowledge about system behavior has been encoded in the parser's knowledge base.

5.1 Components of the Natural Language Interface

To understand the specification of digital systems in restricted English text requires:

1. a corpus (a collection of writings, in this case examples) for the domain of these specifications,
2. a representation for the knowledge expressed in the corpus,
3. a formal representation for the behavior of digital systems, and
4. a parsing technique to map the natural language into the formal representation.

Each of these will now be described briefly.

5.2 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.*

5.2.1 The Corpus' Knowledge

The representation of the knowledge expressed in this corpus was constrained by the choice of a pre-existing 'parsing' technique which was implemented by Arens in PHRAN, a PHRasal ANalysis program [43].

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

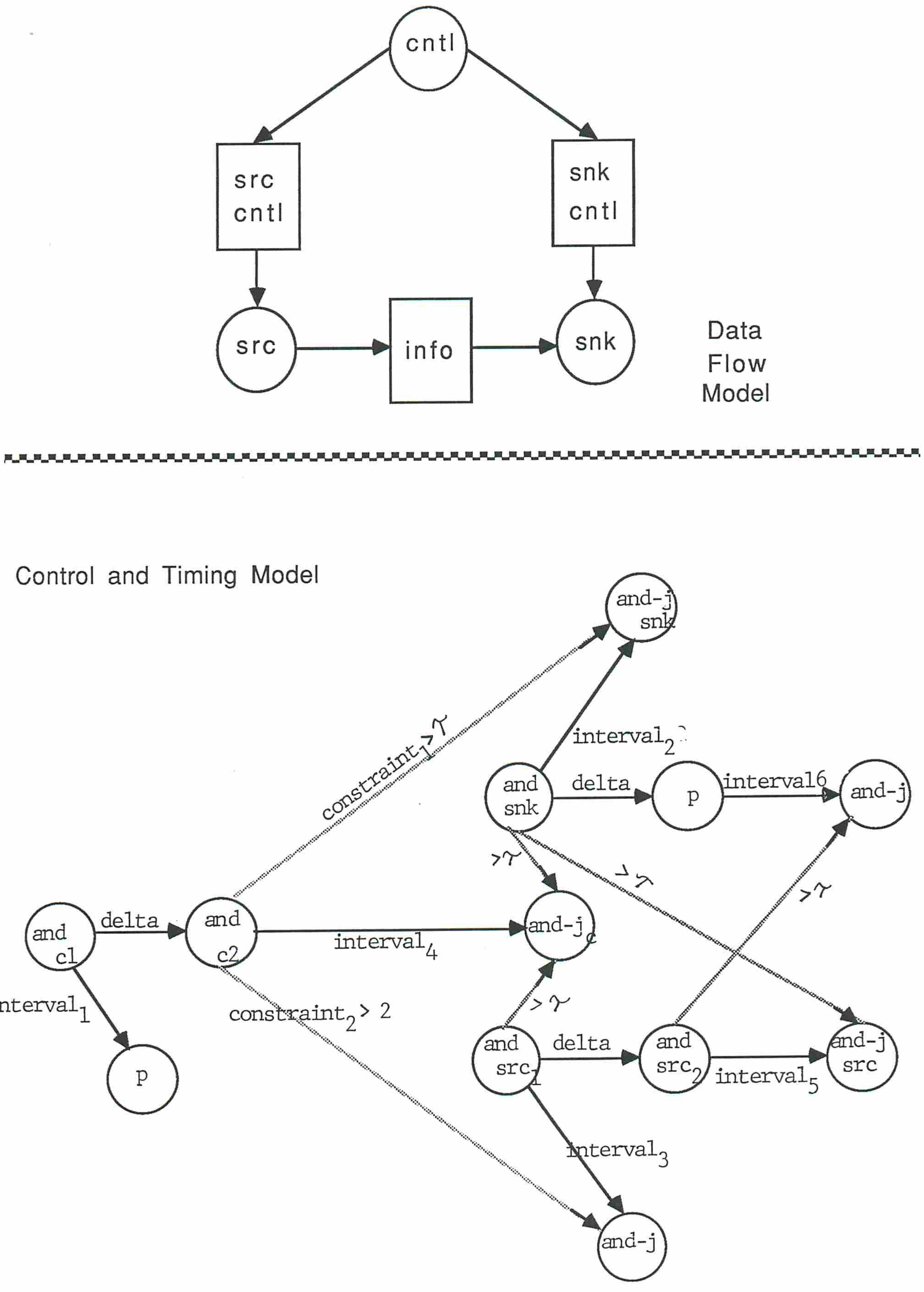

Figure 12: The DDS Template for a UVT.

<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>

is a **conceptual template** that denotes a transfer of data from one component to another physical component.

The concepts in PHRAN are expressed in a specification representation language (SRL) based on Conceptual Dependencies (CDs) as developed by Schank [44]. CDs are a declarative representation of meaning which are based on general concepts of human action, human interaction and other generalizations about physical objects. Our SRL is based on concepts of system behavior and the information required to specify a digital system.

5.3 Representation of Digital System Behavior

Three examples of digital system behavior represented in the DDS will now be given.

5.3.1 The Unidirectional Value Transfer

An example conceptual template is the DDS template for a Unidrectional Value Transfer (UVT) shown in Figure 12 and Table 1.

This template spans two of the DDS models, the data flow model and the control and timing model. The template for the UVT in the data flow subspace 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 value **info** originates or the **sink** operation, the destination for **info** or the **control** may be associated with a third independent operation.

An example of a sentence which maps into a UVT is

Value	range	operation	range
src cntl	interval 4	cntl	interval 1
snk cntl	interval 4	src	interval 3
info	interval 5	snk	interval 2

Table 1: The DDS Bindings for the UVT.

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 control and timing template shown in Figure 5.3.1 is used as the default. The default control and timing template shows the timing for the three operations and the necessary constraints for a valid UVT. For example, $constraint_1$ and $constraint_2$ 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 this constraint 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 the constraint arcs emanate from a node labeled and_2 indicates that the two constraints, $constraint_1$ and $constraint_2$ and the interval labeled $interval_4$, all begin concurrently.

5.3.2 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 Relation (STR) being added to the template. A constraint arc would be added from initiation of the UVT at *and_1* to the termination of $interval_3$, bounding the total duration of the UVT.

5.3.3 The Asynchronous Temporal Activity

The previous example makes no assumptions regarding synchronization or asynchronous behavior. To specify this type of information, the sentence could be prefixed with the adverbial phrase **upon receipt of the flag !b.** [4]

This would create an Asynchronous Temporal Activity (ATA) that would specify the asynchronous branching model. The model for asynchrony is built from the **or**-node DDS primitive which expresses an 'or-fork'.

To accomplish this concept of 'values changing' in a single assignment data flow language a **carrier** from the structural subspace must be introduced to model a changing 'signal' with two values. That is the flag !b may be considered a signal with the value true or the value false and the change from true to false could be represented by binding these values to the carrier in two different intervals.

5.4 The Parsing Technique: PHRAN operation

PHRAN reads the sentence from left to right one word at a time. As each word is examined, existing patterns and concepts are checked for a match and retained, modified or discarded. The match may be based on lexical criteria, semantic criteria and/or syntactic criteria. PHRAN also provides some degree of look-ahead in the sentence to the next word and the ability to look back at previously matched terms with some limited ability to modify those previously matched terms.

The patterns and concepts for PHRAN are stored in a knowledge-base of pattern-concept pairs (PCP). An example of the pattern for the UVT concept expressed by the verb **transfer** is

[(or (a_component) (df_opn)) (root %transfer) (d_val)].

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, which are differentiated 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 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

[4] A **!** prefix indicates a user-supplied label.

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 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

```
(uni_dir_vtrans
        (source (a_component *unspecified*))
        (sink (a_component *unspecified*))
        (info (df_val command1))
        (control (a_component cpu1)))
```

Note the slots for the source and sink have defaulted to ***unspecified*** since they were not included in this sentence.

The source and sink are specified by adding two adverbial phrases to the sentence. These phrases must match the patterns associated with the desired concepts. The patterns are

```
[to (or (a_component) (df_opn))]
[from (or (a_component) (df_opn))]
```

When these patterns are matched the concept associated with each of them modifies the UVT pattern by replacing the default value of ***unspecified*** with the value of source and sink found in the sentence.

The following sentence results in a completely specified UVT.

The cpu transfers the code word from the controller to the peripheral device.

Another option in the prototype system is for SPAN to display the resulting concept in English instead of the frame like data structure.

SPAN's output is the following:

This sentence resulted in a data flow subgraph for a unidirectional value transfer.
The source of the information is the *controller1.*
The sink for the information is the *peripheral-device1.*
The information transferred is the *code-word1.*
The transfer is controlled by the *cpu1.*

6 Current Status and Conclusions

The system currently recognizes simple sentences associated with all the primitive concepts of our specification language which are required to describe behavior in the domain of digital systems. At the time of this writing (October 89), actual pattern-concept pairs have been built for 88 basic verb patterns common to specifications and 468 nouns. In addition, the system uses several hundred of PHRAN's patterns as supplied from Berkeley. Some auxiliary verb constructs for verbs like **shall** have been added to the pattern-concept pairs and the system now has 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. A database has been added so that sentences can be passed in context. Finally new patterns, a concept and new vocabulary were added to deal with loops and iterations.

The system is coded in Franz Lisp and is running in interpreted mode on a SUN workstation under SUN's operating system, Version 1.4 (UNIX BSD 4.2). Typical sentences take approximately 10 to 25 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.

In conclusion, a small set of concepts for system-level specification, including temporal information, 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.

The system also has demonstrated the application of PHRAN to another domain and for a different purpose than previous applications. The work presented here is a step toward using natural language to solve a problem involved with generating information rather than simply asking queries about an existing body of information.

7 Acknowledgements

This research was supported by the National Science Foundation under computer engineering grant #ECS-8310744 by the Departments of the Army, Airforce and Navy under contract #N00039-87-0194 and John Granacki was partially funded by the Hughes Aircraft Co.

References

[1] J. Granacki, 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, Las Vegas, NV*, pages 727–730, June 1985.

[2] Rajiv Jain, Kayhan Küçükçakar, Mitchell J. Mlinar, and Alice C. Parker. Experience with the ADAM system. In *Proceedings of the 26th Design Automation Conference*, pages 56–61, IEEE and ACM, June 1989.

[3] M. Breuer and X. Zhu. A knowledge based system for selecting a test methodology for a PLA. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference, Las Vegas, NV*, pages 259–265, June 1985.

[4] R. Jain, K. Kucukcakar, M. J. Mlinar, and A. C. Parker. Experience with the ADAM Synthesis System. In *Proceedings of the 26th Design Automation Conference*, ACM/IEEE, June 1989.

[5] D. Knapp and A. Parker. A design utility manager. *The 23rd ACM/IEEE Design Automation Conference, Las vegas, NV*, June 1986.

[6] D. Knapp, J. Granacki, and A. C. Parker. An expert synthesis system. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 419–424, ACM-IEEE, September 1983.

[7] J. J. Granacki, Jr. *Understanding Digital Systems Specifications Written in Natural Language.* PhD thesis, University of Southern California, November 1986.

[8] A. Parker, N. Park, and D. Knapp. *Simulation Effectiveness and Design Verification.* Technical Report DISC/84-2, Department of EE-Systems, University of Southern California, October 1984.

[9] A. Parker, F. Kurdahi, and M. Mlinar. A general methodology for synthesis and verification of register transfer designs. In *Proceedings of the 21st Design Automation Conference*, ACM SIGDA, IEEE Computer Society, June 1984.

[10] D. Knapp. Synthesis from partial structure. 1988.

[11] N. Rescher and A. Urquart. *Temporal Logic.* Springer-Verlag, New York - Wein, 1971.

[12] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, October 1977.

[13] B.T. Hailpern and S.S. Owicki. *Verifying Network Protocols Using Temporal Logic.* Technical Report 192, Stanford University, June 1980.

[14] G.V. Bochmann. Hardware specification with temoral logic: an example. *IEEE Transactions on Computers*, C-31(3):223–231, March 1982.

[15] L. Lamport. What good is temporal logic. In R.E.A. Mason, editor, *Information Processing 83*, pages 657–658, 1983.

[16] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. In T. Uehara and M. Barbacci, editors, *Computer Hardware Description Languages and their Applications*, pages 79–90, North-Holland Publishing Company, 1983.

[17] M. Fujita, H. Tanaka, and T. Moto-oka. Logic design assistance with temporal logic. In C.J. Koomen and T. Moto-oka, editors, *Computer Hardware Description Languages and their applications*, pages 129–138, August 1985. Tokyo, Japan.

[18] R.L. Schwartz, P.M. Melliar-Smith, and Fredrich Vogt. An interval-based temporal logic. In *Logic of Programs Workshop*, pages 443–457, 1983. Lecture Notes in Computer Science 164.

[19] Y. Shoham. *Reasoning About Change. Time and Causation from the Standpoint of Artificial Intelligence.* The MIT Press, Cambridge, Massachusetts, London, England, 1988.

[20] M. Rayner. Did Newton solve the “extended prediction problem ?”. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 381–385, 1989.

[21] T.L. Dean and D.V. McDermott. Temporal data base management. *Artificial Intelligence*, 32(1):1–55, 1987.

[22] A. Sathia, M. Fox, and M. Greenberg. Representation of activity knowledge for project management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(5):531–552, September 1985.

[23] A. Bolour, T.L. Anderson, L.J. Dekeyse, and H.K.T. Wong. The role of time in information processing: a survey. *SIGART Newsletter*, 80(80):28–48, April 1982.

[24] J.F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, November 1983.

[25] J.F. Allen. General theory of action and time. *Artificial Intelligence*, 23(2):123–159, 1984.

[26] J.F. Allen and P.J. Hayes. A common-sense theory of time. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 528–531, 1985.

[27] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of AAAI'86*, pages 377–382, American Association for Artificial Intelligence, Washington, D.C., 1986.

[28] R.E. Valdes-Perez. The satisfiability of temporal constraint networks. In *Proceedings of AAAI'87*, pages 256–260, American Association for Artificial Intelligence, Washington, D.C., 1987.

[29] R. Dechter, I. Meiri, and J. Pearl. Temporal contraint networks. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, May 1989.

[30] J.A.G.M. Koomen. Localizing temporal constraint propagation. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 198–202, May 1989.

[31] R. Balzer. A 15 year perspective on automatic programming (invited paper). *IEEE Transactions on Software Engineering*, SE-11(11):1257–1268, November 1985.

[32] K.C. Mander and S.G. Presland. *An Introduction to Specification Analysis - SPAN.* Technical Report CSS/79/12, Department of Computational and Statistical Science - University of Liverpool (gt. Brit.), 1979.

[33] R.J. Abbott. Program description by informal English descriptions. *Communications of the ACM*, 26(31):882–894, November 1983.

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

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

[36] M.R. Grinberg. *A Knowledge-Based Design Environment for Digital Electronics.* PhD thesis, Computer Science, University of Maryland, September 1980.

[37] R. Wilenski, Y. Arens, and D. Chin. Talking to UNIX in English: an overview of UC. *Communications of the ACM*, 27(6):, June 1984.

[38] T. Samad and S.W. Director. Toward a natural language interface for CAD. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference, Las Vegas, NV*, pages 2–8, June 1985.

[39] D. Knapp and A. Parker. *A Data Structure for VLSI Synthesis and Verification.* Technical Report, Digital Integrated Systems Center, Dept. of EE-Systems, University of Southern California, October 1983.

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

[41] M.E. Conway. A multiprocessor system design. In *AFIPS, Volume 24, Proceedings of the Fall Joint Computer Conference*, pages 139–146, 1963.

[42] E.W. Dijkstra. Cooperating sequential processes. In *Programming Languages*, pages 43–112, Academic Press, London, 1968.

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

[44] R.C. Schank. *Conceptual Information Processing.* Elsevier Publishing Company, New York, N.Y., 1985.