CHAPTER 4

CONTENT DETERMINATION

4.1 INTRODUCTION

To generate natural language text one has to perform three tasks Content determination, Planning and Realization (Jacobs P.S 1987, Andersen and Munch K.H.1991). Content determination concerns with what information to include to achieve the particular communicative goal and is a crucial phase of NLG. World knowledge and commonsense knowledge along with the domain knowledge enhance the quality of the generated natural language text. Knowledge of the generation task guides the choice of what information to include (Terry Winograd 1983). Content determination requires huge amount of domain dependent data. In this work, an application has been selected which has a marginally close-ended domain. This restricts the inputs needed for generating a natural language text effectively for achieving the particular communicative goal. In order to facilitate easy access to information the entire knowledge source is needed for identifying the input and the input components so identified should have a uniform representation (Lucja M, Iwanska et al 2000, Devanbu P, Brachman 1990). Since information is conceptual, frames, a knowledge representation scheme, which possesses good inferencing capability, is used for representation. Knowledge sources include domain knowledge, world knowledge and commonsense knowledge (M.Vossers 1991). To generate text without deterioration some of the input components such as concepts, entities, relations, structure, sub process, process, states, transitions, events,
triggers, time, operations, source, destination, results, and roles are identified. Generally, inputs needed for documenting a project, are obtained from various software requirement specification techniques such as data flow diagrams, entity relationship diagrams, state transition diagrams and control flow diagrams. In this chapter, the use of grammars with its limited application of knowledge for generation has been discussed. The use of various knowledge representation schemes for Natural Language Generation has been next discussed. In this work frames has been used as the representation scheme to represent the hierarchical organization of concepts, entities and processes and the relations between them. The general concept of frames and modification to the existing frame structure to cater to the specific task of Automatic Software Development has been discussed in detail. In addition to representing conceptual hierarchy the application chosen in this work needs the representation of causal linkage between the components in the knowledge representation.

Once the inputs are identified, planning is performed. Planner deals with how to organize the identified content. The realization phase then constructs coherent natural language text to achieve communicative goal.

4.2 GRAMMARS AND GENERATION

Basically grammars are used for natural language understanding. Grammars such as Context Free Grammar, Augmented Transition Network Grammar, Transformational grammar, Systematic Grammar, Semantic Grammar and Case Grammar have also tackled generation of natural language. LIFER (Language interface facility with ellipses and recursion), a Natural language generation system, built using semantic grammar has been used for
spelling corrections and processing elliptical inputs. The problem of using ATN is that it forces a certain ordering of decisions about the surface structure of the sentence that must be followed by the system. LUNAR, a system that produces responses to questions has been developed using ATN. SHRDLU is a system that has been developed using systematic grammar, which generates text at the surface level. At the surface level, lexicalization plays an important role. Apart from grammars used for generation knowledge representation are also used for generating natural language text effectively.

4.3 KNOWLEDGE REPRESENTATION IN THE CONTEXT OF NATURAL LANGUAGE GENERATION

Grammar approaches to natural language generation deals with limited application of knowledge and hence lacks effective inferencing (Uma G.V., Geetha T.V. 2002). To generate a natural language text, rich knowledge sources are needed. To access a knowledge base, an effective knowledge representation scheme is required. A representation and reasoning system can be judged on two different grounds. It is epistemologically adequate if it can express the concepts and relations needed to solve the problem. It is heuristically adequate if it can use the information expressed with reasonable computational resources (John F. Sowa 1991, 1999). Various knowledge representation schemes that have been used for natural language generation are as follows: knowledge representation language, using which, Brachman developed a system called KLONE, which supports reasoning based on structured inheritance network. KRYPTON is another system, which focuses on a functional specification of knowledge base. Semantic networks proposed by Quillion, composed of various kinds of association links, nodes and closely reflected the organization of an ordinary dictionary. Simmon’s network
focused on the understanding and generation of particular sentences. A semantic network attempts to combine a factual knowledge and model the associative connections. C. Paris, used semantic networks for building interactive support tool for writing multilingual manuals. McKeown and Moore have developed a text generation based on explanations, using conceptual graphs. By applying inference knowledge to dependency graph J. Moller has built domain related focus shifting constraints in dialogues with knowledge-based systems. Frames, proposed by Minsky, are a complex data structure representing a stereotypical situation. A frame can be viewed as a network of nodes and relations (Minsky 1998, Mark Stefik 1995). CYC, developed by Lenat, attempts to assemble a massive knowledge base (on the order of $10^8$ axioms) using frame based language embedded in a more expressive predicate logic framework along with the features for representing defaults and this has been used for natural language understanding.

As compared to knowledge based approaches to natural language generation there are systems, which are embedded into application-oriented specifics. These include Moore’s embedded menus for patient education application, which dynamically generates follow up questions. YAG (Yet another generator) developed by McCoy is capable of tailoring text based on the feature set available. IDAS, developed by Mellish is based on hypertext for generating technical documents. It also provides short responses to explicit or implicit user queries in the form of hypertext nodes which offer possible follow up to be selected with a mouse. Xu Wu et al. developed a knowledge based database assistant with a query guiding facility.

Natural language generation requires all possible inferences and adaptability. Frame representation has the main disadvantage of occupying
large amount of space during information retrieval, because, to generate even a simple text, a number of links need to be accessed. As the number of links increases, the spatio temporal complexities increase. To avoid such problems and to achieve more adaptability in frames, this work attempts to improve the epistemological status of the frame by including a new component called perspective descriptor. The perspective descriptor will store perspective information about the entire problem. Normally, whenever any change has to be incorporated with respect to domain, the entire frame structure has to be modified. Instead, if perspective descriptor is used, based on the modification, it is sufficient if the corresponding perspective descriptor alone is changed.

4.4 CONCEPT OF FRAMES

Frame is Marvin Minsky's word for concept or schema. Frames are typically arranged in a taxonomic hierarchy in which each frame is linked to one (or in some systems, more than one) parent frame. A parent of a frame A represents a more general concept than does A (a superset of the set represented by A), and a child of A represents a more specific concept than does A. A collection of frames in one or more inheritance hierarchies is a knowledge base (KB). Frames have components called slots (Richard Frost 1998, Patrick J. Hayes 1995). The slots of a frame describe attributes or properties of the object represented by that frame, and can also describe binary relations between that frame and another frame. In addition to storing values, slots also contain restrictions on their allowable values. Slot definitions often have other components in addition to the slot name, value, and value restriction, such as the name of a procedure than can be used to compute the value of the slot, and a justification (in the truth maintenance sense) of how a slot value was computed. These different components of a slot are called its facets.
Inheritance causes slot definitions to propagate down the taxonomic hierarchy. Inheritance is a tremendously useful tool for engineering complex knowledge bases. When a user creates a new frame and that frame inherits slots from its parent, the inherited slots form a template that guides the user in filling in knowledge about the new concept. Because all slot and facet information is available at run time, it is accessible to a program such as a user interface that guides the user in entering new knowledge. For example, the user interface can directly determine the slot data type and value restrictions. Inheritance also facilitates systematic changes to complex knowledge. Some frame representation system compute a relation between class frames called subsumption that allows the frame representation system to automatically determine the correct position of a class in a taxonomic hierarchy (to classify the class). Frame A subsumes Frame B if A defines a more general concept than does B; meaning that every instance of the concept B is an instance of A.

The uses of frame-based systems are inference, by inheritance, classification, by matching, explanation, by matching and inheritance and problem solving, by matching and inheritance. In general, an instance frame describes individual things, a class frame describes entire class, and slots are parts of a frame that contain slot values that describe the object or class. For classes, the values are defaults, which hold typically but universally. In principle, all the information in frames could be translated into rules or predicate calculus (formal logic), but frames naturally support different kinds of procedures. Procedures generally contain inference conditions and rules. Generally following ways can create frames: Abstracting the common slots and values in the given two or more instance frames, creates the required frames. These slots and values then belong to the new class frame. Given two class frames, a new class frame that has all the slots and values that they each have, is
created. Apart from creation using frames, inference can also be done by the following ways.

**Inference by Inheritance**

When a query about an instance or class is to be answered, the is-a and a-kind-of hierarchy is moved up until a frame is found with a slot relevant to the query. Then the default slot value provides an answer to the query. This inference could be made by a rule-based system, but inheritance may provide a much more efficient way of answering the question. If the frame already has a filled slot, then its value is returned.

**Classification by matching**

When an instance is to be classified whose a-kind-of slot needs filling in, the instance is matched against relevant class frames to see which has the best fit. An instance frame is classified by determining which class frame it best matches. The match between two frames (a "hit") is calculated by checking whether both the frames have the same slot with the same value.

Various matching algorithms are possible:

- Degree of match = hits, or
- Degree of match = hits - misses, or
- Consider "near-hits", i.e. same slot but different value

Return the class frame that is the best match, and supplement the instance frame by adding a new IS-A slot with the returned value.
Problem solving

A problem is represented as an instance frame with slots for starting conditions and goals. A match is made between the problem against class frames that describe typical problems and slots describing the solutions. The problem frame (schema) that has the best match is found. Then the problem solution can be obtained by inheritance from the relevant problem schema. Thus problem solving can be viewed as a combination of classification by matching and inference by inheritance, not (as in rule-based systems) as a search through a space of operators.

Explanation

Similarly, a problem is represented as an instance frame with slots for available knowledge and what is to be explained. The problem against class frames that describe typical explanations and have slots describing their solutions is matched. The explanation frame (schema) that has the best match is found. Then the explanation can be produced by inheritance from the relevant explanation schema.

4.4.1 Functional View of Frame Representation Systems

Most frame representation systems provide a library of functions that application programs can call to perform actions such as: adding a new value to a slot, deleting one or more values of a slot, retrieving the current value of a slot, creating a new frame with specified parents, changing the parents of a frame, deleting a frame, renaming a frame, adding a new slot to a class, and adding a facet to a slot (Brachman R and Levesque H 1985, Brachman R. J et al
1995). In addition to the function call library, some systems allow the user to accomplish these same functions with a graphical user interface. For example, KEE, STROBE, CYCL, and KREME allow the user to create graphical displays on a workstation of both the taxonomic hierarchy of a knowledge base, and of the slots within a given frame. Items within these displays are mouse sensitive, and can be used to call up menus from which the operations described in the previous paragraph can be selected. In an approach pioneered by KRYPTON, some frame representation systems provide a declarative language that users can employ to both query a knowledge base, and to assert new facts into a Knowledge base. In PROTEUS, the query would return a list of assertions describing all children. In this work the entities concepts and processes are hierarchically organized with level of detail and numbering according to the level number of the Data Flow Diagram. Hence special library functions are required to delete, components from higher levels of Data Flow Diagram that are represented lower down in the frame hierarchy.

4.4.2 Problem Solving and Inference

Application programs that interact with a frame representation system typically employ either production rules or classification to perform inference based on knowledge stored in the frame representation system. While most frame representation systems provide either production rules or classification, whereas LOOM and CLASSIC support both. In KEE and CLASS, each production rule is itself encoded as a single frame. In KEE and PROTEUS, queries such as those described in the previous section can invoke a backward chaining production rule interpreter to derive the queried slot value. Similarly, THEO users can attach PROLOG rules to slots to cause THEO to backward chain to derive queried slot values. KEE and PROTEUS can also
invoke forward chaining when new slot values are asserted. Classification is used to support inference in two different ways. First, the very act of classification can be a problem solving action, for example, if a system can recognize a description of a patient as an instance of a disease class, it has computed a diagnosis. Second, in the KLONE family of frame representation systems, classification is a key component of the query processor that allows the system to reason about relationships among terms used in a query and terms in a knowledge base (Karp P.D, Myers K 1995). These systems answer a query by translating the query into a concept description, and then classifying that concept to determine its placement in the taxonomic hierarchy. All concepts below the query concept in the hierarchy are subsumed by the query, and thus comprise the answer to the query. The important principle of knowledge representation that has thus far been discovered concerns the complexity of computing subsumption (and therefore, classification). Researchers have compared the cost of computing subsumption in a number of different frame representation system representation languages, and have found that the more expressive the language, the higher the cost of computing subsumption within that language. Some frame representation systems leverage their inference capabilities by combining them with context mechanisms and truth maintenance systems. These facilities are valuable for investigating alternative problem solutions in parallel, and for tracking the dependence of problem solutions on underlying assumptions. Context mechanisms exist in THEO, KEE, STROBE, CYCL, SRL, LOOM, and CRL; truth maintenance systems are present in THEO, KEE, CYCL, LOOM, CLASSIC, KLTWO, and PROTEUS. In the Automatic Software Documentation process the knowledge from the frames need to be extracted during the planning phase. This is done using the classification methodology.
4.4.3 Design principles for knowledge representation systems

The large size of the frame representation system design space implies that the designer of a frame representation system must make many decisions. For example, a decision must be made on what model of the frame and the slot to utilize, what inheritance mechanism(s) to use, whether to employ classification, and what subsumption algorithm to use if classification is employed. A comprehensive set of principles of knowledge representation should guide frame representation system designers and users through a complex web of choices. Frame representation system users need to know what combination of representational constructs will allow them to quickly build an application that has acceptable performance. They need to know what representational constructs can encode the knowledge in their domain most naturally and succinctly, to yield an application that can be maintained easily as it evolves. Users need to know the theoretical costs and benefits of frame representation system features, and they need to know how a particular frame representation system will perform under the demands of their application. Knowledge representation principles should guide users in choosing the optimal frame representation system for a particular problem --- the one with the maximum benefits and the minimum costs. When designing a frame representation system to solve one or more classes of application problems, implementers must address a superset of these issues. As well as anticipating what combination of representational constructs will yield sufficient expressiveness and performance for the applications, the implementers must make a number of engineering decisions. For example, implementers must decide among alternative implementation strategies for the representational constructs they have chosen. Comprehensive frame representation system design principles are largely lacking. The main principle of knowledge
representation that has thus far been elucidated is the expressiveness--tractability tradeoff that relates the expressiveness of a representation language with the cost of computing classification within that language. This principle is clearly valuable since it helps users and implementers understand the expressive benefits and the worst case computational costs of several representation languages. However, this principle describes only the worst-case theoretical impact of one class of representational constructs (concept definition constructs) on one type of frame representation system operation (classification). Many additional principles are needed to cover other representational constructs, other frame representation system operations, other theoretical performance besides worst case, actual performance in addition to theoretical performance, and other criteria besides performance and expressiveness. More specifically: Classification is only one of many operations that frame representation systems compute. Some frame representation systems do not even compute classification. Expressiveness--tractability analyses have not considered representational constructs such as metaclasses, facets, and inheritance across multiple links. Although not all of these constructs will affect the classification operation, they will certainly impact some frame representation system operations. Worst case theoretical results are not always representative of the average case, and theoretical results are not always constraining in practice. Theoretical principles concerning average case behavior of various frame representation system operations are generally lacking. Engineering principles concerning choices of data structures and algorithms are even fewer. Performance and expressiveness are not the only factors to consider when choosing a representation. Two representations might have equal expressive power, but one might be much more succinct for a particular application. In addition, there is a tradeoff between the runtime flexibility of the frame representation system (the degree to which knowledge
that the frame representation system maintains can be altered at run time as opposed to the time of definition), and the performance of the frame representation system. Also, the modularity of a frame representation system implementation is affected by the choice of representational constructs and the implementation of those constructs. Other factors concern the effort involved in implementing a particular feature, and the frequency with which that feature is used in different applications, a feature that is difficult to implement but that is hardly ever used should probably be disregarded. Generally, expressiveness can be achieved by incorporating effective causal links. Because, in the existing frame structure, hierarchy alone can be obtained. By using causal links the entire cause and effects can be understood. While considering about the performance of the frame structure, even if additional knowledge is used or if the domain is slightly changed, reorganization of the entire frame structure need not be done. This is achieved by using perspective descriptors, an additional slot in the existing frame structure.

Slot Units

Several frame representation systems contain a type of frame that the authors of CYCL call a slotUnit. A slotUnit is a frame that holds definitional information about a single slot that describes the use of that slot throughout a knowledge base. A slot Unit might specify the domain and range of a slot S, and the range of this slot describes its allowable values slot Units to store inverse definitions.

In general, a frame is a structured object consisting of a set of slots each of which has one or more of the following items: Procedure to access the value of the slot, Procedure to store the value of the slot and procedure to
remove the value from the slot, constraints, default values and commonsense knowledge depriving the source of value. An important feature of frame is that they can be organized into hierarchies. To achieve effective modularity, frame representation system allows one to describe a set of mappings for all instances of a class. Such a description is called slot value inheritance. To describe inheritance, a template slot is used. Template slot is a description of a slot, which is associated with a class. Each slot possesses value type, cardinality, inverse and collection type. When the slots are filled they are instantiated and hence represents a particular entity of the type represented by the unfilled frame. Frames may be linked to other frames in various ways: (1) A slot in one frame might be filled by another frame (2) frames may be linked in a taxonomical structures, such links between frames can be used to speed up matching and for the inference of generic properties.

So, frames have been chosen as the basic knowledge representation mechanism for generating natural language text in this work.

4.5 KNOWLEDGE REPRESENTATION FOR AUTOMATIC SOFTWARE DOCUMENTATION

The following figure 4.1 shows the entire knowledge representation used for automating software document using frames.
Figure 4.1 Knowledge representation for automatic software document generation
Most of the knowledge base program analysis and understanding approaches produce program documentation, which is generally in the form of structured natural language text. Such informal documentation expresses an intuitive description of the code. However there is no semantic basis that makes it possible to determine whether or not documentation has desired meaning. This lack of firm semantic basis makes informal natural language documentation inherently ambiguous. Documenting user manuals is a very difficult task in software development. Inputs required for any software application can be obtained from data flow diagrams, entity relationship diagrams, state transition diagrams and decision trees. Input components such as entities, actions, process and relation can be obtained through any of the software requirement specification techniques. The identified inputs should be shared with various types of knowledge sources such as (1) domain knowledge, which consists of background knowledge, technical knowledge, additional knowledge and structural knowledge (2) commonsense knowledge and (3) linguistic knowledge that constitutes semantic knowledge and syntactic knowledge. In the work described in this thesis the entire knowledge is represented using the frame structure. Frames in general can be classified into four types of frames such as syntactic frame, semantic frame, thematic frame and narrative frames. Thematic frames and narrative frames are used for content determination. Semantic frame are used for planning and syntactic frame for serialization. Frames can be represented using first order logic.

Fuzzy logic can be used to represent inexact matching. P.D. Karp has extended the features of frame system by providing high-speed access for retrieval of information. He has used knowledge base by including all the facilities of database management systems. The existing frame structure has been modified in order to access quickly and to reduce spatio temporal complexities.
4.6 FRAMES EXTENDED WITH PERSPECTIVE DESCRIPTOR

Frames are well suited, for representing knowledge efficiently even if the entire domain is not explicitly specified. That is, any domain for natural language generation, there may be focus shifts within the domain itself, where, the actual knowledge about the entire domain cannot be predetermined. In the case of automating a software document one has to generate user manuals based on the required levels since various levels of manuals have to be generated depending upon the status of the user. Hence knowledge of the domain is not predetermined and completely specified until the level of document to be generated is specified.

In this work a single frame represents an entity or an action. Sequence of operations and actions are represented by a number of frames. A frame representing an entity or action can be part of many different operations. As the level of details of an entity for an operation is different for different levels of documentation, the connections and the knowledge associated with a frame representing an entity or action has to be dynamic in nature. This in effect means that the complete representation of the domain has to be dynamically reoriented to adapt to the type of operation and level of details required. Moreover in the case of automatic software documentation, which is considered here, there will be a need, to be able to incorporate modification to be document whenever there are minor modifications to the domain of operation of the software or when there are small enhancements to the software. Thus it is imperative that a knowledge representation methodology used for software documentation should be able to also incorporate these modifications without drastically recreating the complete knowledge representation. Hence the knowledge represented cannot be static. To achieve this dynamic capability, adaptability and flexibility a new component called perspective descriptor has been added to the existing frame structure.
The perspective descriptor is used to provide flexibility in the knowledge representation even when the domain is slightly modified (Uma G.V. Geetha T.V., 2001). Each frame can have \( n \) number of perspective descriptors \( p_{d1} \ldots p_{dn} \) each describing individual persecutions about the domain. Each and every perspective descriptor is responsible for producing \( m \) number of perspective concepts. The perspective descriptor will identify only the domain change, and each change/operation is associated with various concepts. Perspective knowledge about the descriptor will produce the corresponding perspective concepts.

The following figure 4.2 shows how a frame with a perspective descriptor is defined.
For any domain D the perspective descriptor is represented as < D, Pd, Pc > where D represents the domain, Pd refers the perspective descriptors and Pc, perspective concepts

4.6.1 REPRESENTATION OF PERSPECTIVE DESCRIPTORS AND PERSPECTIVE CONCEPTS

Based on the concept perspective analysis is performed about a particular domain. That is, the various different types of operations/actions to be performed for the domain are analyzed. Using the zeroeth level DFD and the world knowledge, the perspective analysis is performed. The results obtained from the analysis are called perspective descriptions. Apart from the DFD and world knowledge and obtain the user’s perspective about the domain, based on common sense knowledge. These are organized into a hierarchy in which a concept is connected to its super concept by an ISA link. This is indicated by the schema < C, L, P, F>, where C denotes the concepts L represents the corresponding link associations P represents the properties and F the corresponding functional mapping of concepts.

The domain perspective scheme is composed of a set of perspective descriptors. When a particular perspective descriptor is activated the corresponding perspective concepts that are closely related are activated. To identify a particular pc associated with a pd it is necessary to eliminate all the redundant properties of the concept. For eliminating, a schema map has to be generated which consists of all concepts. Between the concepts obtained from Pd, intersection set has to be identified. The highest ranked element from the intersection set will be the activated Pc between the closely related PC’s. The
domain perspective scheme is defined as $<D_{pers}, f_{link}>$, where $D_{pers}$ represents perspective descriptors and $f_{link}$ represents frame links.

Consider the example of university students database. The following figure 4.3 shows the zeroeth level DFD from which the basic concepts are obtained.

![Figure 4.3 University students database](image)

The relationship and links between entities are obtained from ER diagram. Given a concept $C(L_1 \ldots L_4)$, $L_1 \ldots L_4$ are the established links that makes explicit the relationship between next link $L_{i+1}$ and $C(L_4 \ldots L_4)$. A frame system generally consists of sequence of operations associated with more number of links. Since the domain is not predefined, there can be possibility of minor changes in the operational sequence. To accommodate these type of changes, the number of links will be increased by means of which, the search
capability will be decreased and space and time will be increased. To avoid this situation, frame system is enhanced with the perspective descriptor.

4.6.2 Memory constraints tackled by the perspective descriptor

The number of links used to represent a concept in an ordinary frame structure is large. Each link is associated with assertion about the concept. Due to the necessity of flexibility in the frame structure, perspective descriptors are provided, which reduces the number of links needed to represent an operation / process by eliminating the redundant frame links. The elimination process will be easier if the links and associations are represented as assertions. The following figure shows the data structure for implementing the frame links using pointers.

The ordered pair represents the current frame link and the corresponding next frame where it is connected. Hence the assertions corresponding to the frame link can be represented in the binary form. Suppose if $P$ represents the links corresponding to the concepts, then the assertions are represented as $(P_{a1}, a_2 \ldots a_n)$ which means that can also denote the relationship between two assertions $a_1$ and $a_2$. This is represented as:

$$(\text{arg}_1, r, a_1)$$
$$(\text{arg}_2, r, a_2)$$

.$$
Here r denotes the relationship. The redundant links are identified from the schema map which, is generated from perspective concepts and they are eliminated. Hence it results is less number of links to represent a concept / operation, which in turn reduces the spatio temporal complexities.

4.7 ALGORITHM FOR IDENTIFYING CORRESPONDING INPUTS NEEDED FOR GENERATING NATURAL LANGUAGE TEXT (SOFTWARE DOCUMENT)

Step 1: Get inputs [from any of the software requirement specification technique such as Dataflow diagram, Entity relationship diagram, state transition diagram] Such as entities, relation, process, sub process etc.

Step 2: Extract inputs according to template designed and fit in the frame i.e fill the appropriate slots.

Step 3: Each frame represents an entity / operations. Hence for each entity {
    Construct a class frame
    Construct an instance frame
    Fill slot values in the slot
    Read value from the slot
}

Step 4: Construct a class precedence list, which produces proper topological sorted order
Step 5: For each action
   do until all frames are exposed
   {
   Construct Information frame model
   Fill in the slot
   }

Step 6: Integrate all the knowledge sources.

Step 7: If any change has to be incorporated, in the domain {
   a. Perform domain perspective analysis
   b. Obtain set of domain perspective descriptors from domain
      concept knowledge and domain perspective knowledge.
   c. Find closely related perspective concepts for a particular
      perspective descriptor
   d. Choose appropriate perspective concept and activate
}

Step 8: Process the identified inputs and pass it to planning phase for
organizing and then the organized content is passed for realization to produce
the automatic documents.

Inputs are obtained from DFDs and they are extracted and
represented in Frames. According to the domain change, perspective descriptors
will be generated based on to perspective analysis of the domain and it activates
the perspective descriptor that has to be modified dynamically. Along with this,
the inputs have to be organized in a causal and conceptual manner.
In this work an attempt has been made to generate documents at different levels with different lexicalizations from the same conceptual representation. The hierarchical structuring of the information allows the extraction of knowledge from the representation at different levels of detail. Moreover as already discussed there is a need to causally link concepts, process and entities. The frame structure has been adapted to allow for these linkages also.

4.8 NEED FOR CAUSAL LINK

Cause analysis has been performed for defect prevention in software the causes of the most prevalent defects are determined. The objective of the cause analysis is to determine the cause of each of the objects, and the major cause categories.

Causal link is required for software documentation, because while documenting any type of document one should infer the domain knowledge using various knowledge sources, that is, what caused what, or what will cause what, and also see why one thing caused another or why one thing will cause another. In software documentation the domain knowledge corresponds to input that are obtained from various SRS techniques. This knowledge has to be represented in a uniform manner in order to obtain precise and consistent information. Knowledge representation can be viewed in terms of information patterns associated with it.

The conceptual links provides the hierarchy of the frames. It is used to integrate the various frames that are related to a particular view. It mainly accepts the related information. On the other hand the causal links provide
causal ordering between the frames. Causality is a logical consequence of the facts of a situation and need the various temporal operations like before, after, during, leads, lags, start, finishes, meets and overlaps. The role of cases such as by (agent), with (co agent), for (beneficiary), from (source), to (destination) is also taken into consideration for the formation of causal link. The general ontology improves the epistemological status of the causal chain. Causal links relate states where each state is an effect of another state that is state of kind K1 at t1, will produce a state of kind K2 at t2. If every instant in the interval k has a cause that still leaves open whether the interval k itself has a cause. If the existence of every member of a collection is explained the existence of the collection is also explained. Based on different views such as informational view, behavioral view and functional description view of software requirement analysis phase, the causal chain is developed.

4.9 INFORMATION FOR CAUSAL LINKS OBTAINED FROM VARIOUS VIEWS

Basically three views are considered for any software documentation. Informational view contains various entities their relationship and attributes presented in the domain that describes data modeling during documentation. Functional description view provides information about process, subprocess, request given to any process, reply etc that describes functional modeling and behavioral view provides information about states, events, actions, triggers, preconditions, post conditions of the given domain, which describes behavioral modeling of the domain during documentation. Behaviors can interact in many ways. For example one behavior can suppress the output of another. The behaviors are organized into a layered hierarchy with lower layers representing less abstract behavior and higher layers representing more abstract behavior.
Information view and functional description view corresponds to technical knowledge of the domain. Behavioral view corresponds to commonsense knowledge. Inputs with respect to non functional requirements such as resources, cost, schedule, effort, hardware and software requirements, configuration management plan, quality assurance plan, breakdown structures, related document, constraints such as technical constraints, operational, environmental corresponds to world knowledge. All these views are used in determining causal linkages between different domain and world knowledge concepts, which will help in enhancing the flow of information required for software documentation.

The logic of causal analysis and the problems involved in establishing causal linkages are associated with three key criteria for inferring a cause and effect relationship (Wallis and Edward H, Shortlife 1984): (a) co-variation between the presumed cause(s) and effect(s); (b) temporal precedence of the cause(s); and (c) exclusion of alternative explanations for cause-effect linkages. Causation relationship frequently depends upon simultaneous, or temporally sequenced, operation of other causes. However, causal inference presents difficulties that must be overcome by any causal induction algorithm. For example, given two related variables X and Y, there are three possible explanations for their relationship. X may cause Y, Y may cause X, or the two may be correlated due to their relationship to a third variable and thus have no causal relationship at all. Another difficulty is that the number of possibilities grows exponentially with the number of variables in the dataset. The result to events in which the output of functionally defined states is causally relevant to objects and to any internal mechanism that is causally relevant to that output (T) is defined.
(T) For any events \( e_1, e_2, e_3 \), event types \( E_1, E_3 \), and circumstance \( C \),

If 1) \( e_1 \) causes \( e_2 \) and \( e_2 \) causes \( e_3 \);
2) \( e_1 \) falls under \( E_1 \) and \( e_3 \) falls under \( E_3 \);
3) \( E_1 \) is causally relevant to \( E_3 \) in \( C \);

To check whether functional properties and states can be causally relevant to the output in terms of which they are defined and further events or states, which are caused by such output, has to be considered.

4.10 CAUSATION AND CAUSAL RELEVANCE

In any software requirement specification technique the causal relation between process, entity and data store can be obtained from the given specifications of the process, sub process, request, and reply, states, events, and entities and data store. Causation and causal relevance are distinct relations. Causation is an extensional relation between events. Causal relations are expressed in singular causal statements. A singular causal statement is of the form \( A \) caused \( B \). By extension, causal relations between objects and states, or objects and states and events. The need to distinguish between the relation of causation and causal relevance arises because in addition to an interest in what caused what, or what will cause what, we also see why one thing caused another or why one thing will cause another.

Causal relevance is of importance because, it concerns the conditions for predicting and controlling events and circumstances, causal relevance indicates that if event types \( E_1 \) and \( E_2 \), relative to circumstances \( C \), are to stand in the causal relevance relation, the statement that an event of type \( E_1 \) occurred,
in circumstances of type C, does not entail that a distinct event of type E2 occurred, or vice versa. Causally committed properties of events (which define an event type) come in two varieties, forward looking and backward looking, which are expressed by predicates of the form: is a cause of F ands caused by F, respectively. An event type E1 is causally relevant to an event type E2 relative to circumstances C only if E1 is strongly logically independent of E2 in C.

**Functionalism and Causal Relevance**

Functionalism and Causal Relevance is related to software documentation because based on functional and non-functional requirements necessary information has to be identified. Consider functional states, characterized in terms of deterministic causal relations. Any deterministic functional state can be characterized by a set of ordered triples of input, output, and future state types. The triples represent what output O (which may be the null output) the system, which has the state, would produce given certain input I (which may be the null input) and what the next state S of the system (which may be the same state) would be. A set of triples of this sort will define or partially define a functional state. A system is in that functional state provided that the set of triples correctly characterizes the counterfactual relations among input, output, and other states of the system. For the functional state to be well defined, if the transition state is itself a functional state, it must be defined as well. The reason a functional state F is not causally relevant to output O in terms of which it is defined is that for it to be causally relevant to O it must be that there are circumstances C in which it is sufficient for O; but given the definition of F, those circumstances must include some input, I, specified in a triple which partially defines F. But if C entails that the system received input I, then C and F are logically sufficient for output of type O to be produced. F thus
fails to meet a necessary condition, Causal Relevance, on standing in the causal relevance relation with O. The same point, of course, applies to the relation between being in functional state F, and coming to be in any of the functional states in terms of transitions to which F is partially defined.

4.11 FUNCTIONAL DESCRIPTIONS OF NON-FUNCTIONAL PROPERTIES

It is important to distinguish between functional descriptions of states or properties and functional definitions of states or properties. The former identify properties or state types by descriptions of their roles in producing output and transitions to other states given certain input, but do not express the properties or types. The latter express the properties or types being defined. Functionally defined properties cannot stand in the causal relevance relation. In contrast, functional descriptions of properties do not preclude the properties described from standing in the causal relevance relation, and may require them to do so. For all x, x is F iff x has the property causally responsible for R in circumstances C. Functional states are defined in terms of counterfactual conditionals. In the characterization of functional states above, each triple corresponds to a conditional of the form: if the system were to receive input I, it would produce output O and move to state S. These requirements in turn show that functionally defined states cannot be causally relevant to the output in terms of which they are defined or to states in terms of transitions to which they are defined. Together with (T), the requirement on the transmission of causal influence, this shows that they cannot be causally relevant to either the mechanisms that produce the output in terms of which they are defined or to that to which that output is in turn causally relevant. Functional definitions are to be distinguished from functional descriptions of properties. Causally relevant
properties can be identified by descriptions of their functional roles, but are not themselves functional properties.

For documenting any type of software causal links need the various temporal operations like before, after, during, leads, lags, starts, finishes, meets and overlaps. Consider the following example: Representing information about processes. Each process can be described as a partial sequence of sub process. Consider a process P consisting of sequence of steps P1, P2, P3. Another process Q consisting of Q1, Q2 occurring in any order, but not in same time. Let Q2 be decomposed into two sub processes Q21 and Q22 each occurring simultaneously. There may be interactions among sub processes. For instance Add that Q1 before Q21, and infer a new relationship between Q1 and Q2. Because Q1, Q2 share the same reference interval Q, Q2 overlaps P1.

4.12 ORGANIZATION FOR CAUSAL DISCUSSIONS

In general, software document requires the discussion of processes that occur in sequence. This in turn, is dependent on cause and effect relation between the concepts entities and processes. Some of the issues that affect causal progression is given below.

Organization of the contents of a causal discussion depends on how many and what combination of causes and effects occur.

- Single cause-single effect--A single cause can lead to a single effect;
- Multiple causes-single effect--Many different causes can be seen as leading to one effect
• Single cause-multiple effects—A single cause can be seen as producing numerous effects.

• Sequential causes and effects—One cause can bring about an effect, which in turn becomes the cause of another effect, and so on.

• Alternate causes and effects—Causes and effects can be alternating.

Some of the questions to be answered while using causal links are as follows:

• What are (or were) the causes of this? How and why does (or did) this happen? What brought about a situation, problem, or accident?

• What are (were or will be) the effects, results, or consequences of this? What will happen if a certain situation or problem continues?

• How does this work? What causes this to function as it does?

• Why won't this thing work? What's wrong with it?

• What changes will occur if a certain plan or action is taken?

• How can a certain problem or situation be avoided?

• What are the advantages, benefits, or disadvantages of an action or object?

Since many processes are involved in the application, causal link is required in order to identify the sequence in particular.
Systemic-functional Perspectives on ‘Process’ are considered for dealing with causal links.

Halliday states that grammatically a process potentially consists of three components:

- The process itself,
- Participants in the process,
- Circumstances associated with the process.

A verbal group typically realizes the process itself. Entities classified under process can usually be expressed as verbs and are frequently the main verb in a clause; this contracts with entities classified under configuration, which would be realized by the clause itself.

Participants in a process typically come from the object hierarchy and are realized as nominal groups, although there are, obviously, exceptions --- such as processes which relate processes themselves in addition to objects (e.g., causality, or mental processes that describe propositional attitudes, etc.) (Sowa 1984).

Circumstances are usually taken from the circumstance hierarchy and often appear as prepositional phrases (Kirk Ludwig 1998, Kitcher 1989, Skryms 1980, Sober 1987). While the participants of a process are considered to be in some sense essential to the performance, or ‘actualization’, of the process, circumstances provide additional contextualizing information such as temporal and spatial location, manner of performance of the process, purposes, etc. The
precise distribution of participants and circumstances depends on the type of process.

The Actor is engaged in a process; does the process extend beyond the Actor, to some other entity, or not?

Halliday suggests that it is also necessary to provide a complementary analysis in terms of an ergative model where the focus is one of causation rather than of extension. Here the question is:

"Some participant is engaged in a process; is the process brought about by that participant, or by some other entity?

The participants - actor and agent, and goal and medium need not correspond and so both dimensions of organization need to be maintained to obtain maximally adequate analyses. In our descriptions of processes and participants below, and occasionally refer to this interpretation of events; in particular, need to refer to the medium participant in a process as that participant most centrally concerned or affected.

Actee is a process participant describing the entity upon which a process is 'done', 'carried out', etc. The actee role is divided into two subtypes: result and process-range.

These subtypes are not at present explicitly referred to by the inquiry implementations used in PENMAN since they are predictable from the process-type. For example, the actee of a directed-action (which is a type of material-process) can only be of the result type; similarly, the actee of a nondirected-
action can only be of the process-range type. This will be clarified in the figure 4.4 on the material-process sub hierarchy below. For example, let us consider a process called process No.2. If it possesses sub processes 2.1, 2.2, 2.3 then if 2.3 is invoked it is understood that process No.2 is instantiated. Hence, generalized possession property is matched. If sub processes 2.3 is in execution, then intensive property is matched. Similarly, if there is need in 2.3 for a further adaptability, circumstantial property is matched.

This participant is termed in Halliday (1985 a) and in Eggins (1994) as Goal.

\[ \text{Figure 4.4 Material process sub-hierarchy} \]

Causal

This concept corresponds to a Causal configuration, it simply captures the relationship of one thing being the cause of another, the effect. Logically it has to roles: the Cause and the Effect (Zhang N.L. and Poole D 1996).
It can be typically expressed by sentences such as:

'\langle cause\rangle \text{ causes } \langle effect\rangle' \\
'\langle effect\rangle \text{ because of } \langle cause\rangle'

The cause can be further specified using one of the causal-relations

Causal-Relation

As shown in figure 4.5 the sub hierarchy can find relations that express a generic notion of "cause", as the followings:

- reason
- purpose
- concessive
- client

Some of these have further sub-classes that are also classified elsewhere in the

![CAUSAL-RELATION Diagram]

**Figure 4.5 Casual relation**

Upper model language of formal logic, e.g.,
Because A implies B and A is known to be true, B.
since A implies B and A is true, B.
A implies B and A is true, therefore B.

Whether a separate concept is required here or not depends finally on whether or not the grammar makes a distinction between these constructions and those of, for example, reason. Only if it does make a distinction is a separate category motivated according to the definition of the upper model that are followed in this document (Paoloterenziani 1995, Salmon 1984, 1994).

Notice that also one of the participant roles is a subtype of causal relation.

**Generalized-Possession**

The most typical expression of *generalized-possession* is as:

<possessor> has <possessed>.

Relations in this category can, in general, also be expressed with a possessive form. Thus, the types of relationship covered by generalized possession, or generalized marriage as it is sometimes called, are rather more general than simple possession of objects and include social customs or agreements.

<possessor> has <relation>
<possessed> be relation of <possessor>
The conception of possession is thus quite general, as is intended by
the use of the term generalized-possession.

Generalized-possession (as shown in figure 4.6) has five specific
subtypes that are currently used by the grammar: part-whole, ownership,
name-of, Generalized-Role-Relation, Ascription-Inverse.

Clearly these do not exhaustively cover the class of generalized
possession, but they are those with which the grammar is concerned in its
present state of development.

Figure 4.6 Generalized possession

Logical Relations

Logical relations combine processes or states of affairs into larger,
composite processes or states of affairs, either conjunctively (e.g., 'and'), or
disjunctively (e.g., 'or'), or by providing more information in an elaboration.
These possibilities for expression define the three main subclasses of logical:
conjunction, disjunction, and elaboration respectively. If 2.1, 2.2 and 2.3 as
considered as sub processes of process 2, then they are logically related
(Alexander and Davis 1991).
Elaboration further divides into two classes that are used to discriminate between possible inter-clause and inter-nominal group relationships: **restatement**, **exemplification**.

The classes below **logical** are shown in the following figure 4.7.

![Figure 4.7 Logical relation](image)

Scalability: A quality is scalable if an object may possess it to varying degrees (the set of possible values form a continuum). For example, 'heavy' is a scalable quality.

1. Type of contrast: the space of values may be divided in many ways
   - Polar: A quality is polar if it has a corresponding quality describing its opposite or the absence of this quality entirely. A polar quality is not part of a larger taxonomy.
   - Taxonomic: All qualities that are not polar are taxonomic, i.e., the possible values can be listed. For example, the quality of being mammal is part of some taxonomy.
Modal qualities (shown in figure 4.8) are qualities of being able to do something, wanting to do something, having to do something, etc.

They are classified along two dimensions: conditionality and volition. They may be described thus:

- Modal qualities that are not conditional are expressed with modalities such as: “will”, “must”, “can”, etc.; those that are conditional are expressed by “would”, “might”, “could”, etc.
- Modal qualities are classified in terms of the actor’s active decision or volition in the performance of a process: a nonvolitional process is one where the actor did not take, or is not expressed as taking, direct responsibility for the process; a volitional process is one where the actor did take responsibility. Volitional processes are often expressed with: “will”, “won’t”, “would”; nonvolitional with: “may”, “can”, “must”, “might”.

Ownership

This is a relation between the owner of an object and the object. It is a specialization of generalized-possession. It has the same roles as generalized-possession, but the possessor role is value-restricted to active-entity.

Ownership may be expressed as:

<possessor> own <possessed>

or

<possessed> belong to <possessor>.

Part-Whole

This is a relation between an entity and its parts. It has two roles: whole (the domain) and part (the range).

At the most general level, this relation can be expressed as:

<part> be an element of <whole>

or

<part> be a component of <whole>

or, as inherited from the generalized-possession superconcept,

<whole> has <part>

Note that how this relation is expressed in the language seems to depend on the type of object that fills the whole role.

There are three possible subtypes that could add for part-whole, but which are not currently distinguished within the grammar:
• consists-of --- which would be expressed as: <whole> consist of <parts>
<parts> make up <whole>

• constituency --- a specialization of the part-whole relation in which the whole is value-restricted to be a decomposable object.

• The justification for this sub-category is that it only makes sense to discuss "parts" of an object when that object has distinguishable parts, i.e., is decomposable into those parts. So, An engine is a part of a car is acceptable, while Gravel is a part of concrete seems less so.

• ingrediency --- this is a relation which expresses the relation between a whole and its parts when the whole is a mass-object.

• Purpose

Purpose also expresses a volitional effect. However, in this case the cause is considered 'future' with respect to the effect. Expression of this relation uses terms such as: "for" with either a noun or verb, "to"<verb>, "by"<verb>-ing, and "in order to/that".

This relation is useful for any system that needs to represent the causality of behavior; for example, constructions such as:

In order to achieve <goal>, program did <plan>. <Plan> was executed for the purpose of <goal>. Program achieves <goal> by applying <plan> will be common.
4.13 CAUSAL LINK REPRESENTATION FOR AUTOMATIC SOFTWARE DOCUMENTATION

In the illustrated figures 4.9, 4.10, 4.11, 4.12, 4.13, Causal links are shown as dotted lines and conceptual links are shown as thickened lines. The knowledge is represented using frame structure with perspective descriptors. Let us consider the non-functional requirements such as cost, resource, schedule, effort etc. Causation and causal relevance is incorporated in it and the causal link provides various specific knowledge about the domain (Uma G.V. Geetha T.V. 2001).

For example to evaluate effort one has to obtain necessary information in the form of cause and effects and its relevancy has to be checked.

Resource cause cost & schedule cause effort, which is relevant to effort. This can be represented as

\[
\text{Cause (resource, cost),} \\
\text{Cancause (schedule, effort).}
\]

To generate/evaluate glossary then the knowledge has to be obtained from the parent document that can cause related document, and, it leads to applicable document. This applicable document causes reference document that is relevant. Based on the functionality, causation and causal relevance can be checked for each entity. Hence to obtain glossary the effect of all the documents (parent document, applicable document, related document, reference document).
Figure 4.9 Casual and Conceptual Link from Functional Requirements

Casual links are shown as dotted lines and conceptual links are shown as thickened lines.

Pd: Perspective descriptors, Cl: Casual links
Figure 4.10 Casual and Conceptual Link Representation from Non Functional Requirements
Figure 4.11 Casual and Conceptual Link from Constraints
Figure 4.12 Causal and conceptual link from project plan
Figure 4.13 Causal and conceptual link from Information Description
Causal link can be used to associate the corresponding knowledge and specific entities (Halpem 2000, Hanna 1986). Causal links used in the content determination can be used to enhance the knowledge required for organization in planner. Causal links are established between processes and entities represented using frame structure with perspective descriptors, which have been extracted from various views and knowledge sources. The various inputs that are identified provide information about procedural flow along with the corresponding relationship between attributes, processes, subprocesses, events, actions and states. The perspective descriptors decide the conceptual hierarchy. Separate perspective descriptors are also used for causal linking. A process in the software may have to undergo a chain of changes during a certain time interval. A process can also be considered as either an abstract element or as a decomposable element. Entities, which are of conceptually similar, will be grouped together. The use of processes in defining categories of entities provides the relationship existing between the two forms of knowledge, declarative and procedural. The views can be integrated along with knowledge sources in order to provide causal linkages. Since knowledge representation mechanism is involved to represent the inputs along with causal links, uniformity can be achieved. Redundant information will be avoided and it also enhances completeness. As the inputs identified are precise and complete, a complete plan can be generated for organizing the text. Causal links provide easy access of information and, the internal connections between the inputs and knowledge can be obtained. Similarly if causal links are applied to a planner, the corresponding template and standard specific to a particular process/entity can be organized accordingly. The causal linkages and causal relevancy obtained for knowledge representation of the software process helps in producing effective software documentation. These identified inputs have to be organized in a uniform manner. Hence planning has to be performed which organizes the structure in a uniform way of representation.