CHAPTER 1

INTRODUCTION

1.1 NATURAL LANGUAGE PROCESSING

The goal of natural language processing (NLP) is to design and build a computer system that will analyze, understand and generate natural human languages (Michael Zock 1990, 1991, Kukich K 1988, Lee Fedder 1990). In general natural language understanding (NLU) and natural language generation (NLG) are the two important aspects of NLP. Natural language understanding (NLU) is the process of analyzing text and producing a knowledge base, which incorporates the information found in the text. Natural Language Generation (NLG) is the process of automatically constructing a natural language text in order to meet specified communicative goals (McDonald 1994). The problem of communicating with computer systems in natural language has always been a challenging task. In spite of the number of programming languages and graphical user interfaces available to interact with computers, building natural language understanding and generation systems is still an area of current research. This is because natural language has the following advantages:

1.1.1 Advantages of natural language

- Natural language is very expressive and computationally very tractable. There are two sources of Natural language’s tractability. Rich structure and well defined nature. Natural
language is highly structured. There is a well defined close correspondence between syntactic structure of Natural language sentences, their fragments and the semantic relation among them.

- Natural language is general purpose, which when combined with its great expressiveness allows one to achieve a uniformity of representation and reasoning and therefore a simple and powerful system.

- Natural language has a uniform level combining verbal and sensory information. Each non-verbal information (knowledge) can be represented as a Natural language utterance (Winograd T 1983).

- Logical contradiction and logical redundancy play a central role in Natural language. They serve as a means of identifying knowledge gaps and false beliefs and also as a means of conveying systematic non-literal meanings.

- Natural language greatly facilitates learning and knowledge acquisition by providing expressive and tractable mechanism for forming or revising hypothesis and by reducing hypothesis search space and providing a number of easily parsable constructions specialized to convey taxonomic knowledge.

- Natural language mixes object level and meta level descriptions. This is advantageous because the system uses the same representational and inferential mechanism to draw inferences about the environment and to reason about its own knowledge and reasoning. Such systems are easy for people to use. Most human knowledge is encoded and communicated via natural language.
Application areas of natural language processing broadly range from spelling, grammar, style checkers, natural language interfaces, machine translation, automatic generation systems. Applications include:

- **Writers' aids**
  - spelling checkers
  - grammar checkers
  - style checkers
  - on-line monolingual and bilingual dictionaries, and dictionary access systems

- **Information management tools**
  - automatic indexing systems
  - text retrieval systems
  - information retrieval systems with a natural language component
  - authoring aids with an information management component

- **Natural language front ends for**
  - information systems
  - data base systems
  - computer systems

- **Translators' aids**
  - translation memories
  - specialised workstations
  - terminology management data bases
The applications of NLP include machine translation of one human-language text to another; generation of human language text such as fiction, manuals and general descriptions; interfacing to other systems such as databases and robotic systems thus enabling the use of human language type commands and queries; and understanding human language text to provide a summary or to draw conclusions.

In this thesis, we discuss the automatic generation of software documents. In general, the objective of natural language generation (NLG) or text generation systems is to produce coherent natural language texts, which satisfy a set of one or more communicative goals. In order to achieve these goals, the generated text should have the following characteristics:

- Coherency: use well-connected, sensible and comprehensible language
- Accuracy: contain accurate information (or it could lead to the user making false inferences)
- Validity: cause the user to make the desired inferences (for example, telling a naive user that the koala looks like a teddy
bear and not telling her that it doesn't behave like one may result in a nasty surprise)

- Informativeness: present new and interesting information to the user
- Understandability: include information which the user can understand (for example, when describing the accordion to a person who is unfamiliar with it, we should: (i) use non-technical terms, since highly technical terms will lead to confusion; and (ii) take advantage of the hearer's existing knowledge of other entities such as the piano in order to make the description more understandable)
- Relevancy: include information, which is relevant to the current discourse goal and not redundant.

Natural Language Generation is a sub field of Computational Linguistics and language-oriented Artificial Intelligence research devoted to studying and simulating the production of written or spoken discourse. The study of human language generation is a multidisciplinary activity, requiring expertise in areas of linguistics, psychology, engineering and computer science. One of the central goals of natural language generation is to investigate how computer programs can be made to produce high-quality natural language text from computer-internal representations of information. Natural language generation often is characterized as a process that has to start from the communicative goals of the writer or speaker and then employ some sort of planning to progressively convert them into written or spoken words. In this view, the general aims of the language producer are refined into goals that are increasingly linguistic in nature, culminating in low-level goals to produce particular words.
From a theoretical perspective, the process of language generation is viewed as goal driven communication (Ehud Reiter and Robert Dale 1999). In general terms, we can characterize the input to a single invocation of an NLG system as a four tuple $<k, c, u, d>$ where $k$ is the knowledge source to be used, $c$ is the communicative goal to be achieved, $u$ is the user model, and $d$ is the discourse history (Ehud Reiter and Dale 1999). The conventional NLG architecture is given below:

![Diagram of Traditional NLG System architecture]

**Figure 1.1 Traditional NLG System architecture**

Traditionally, NLG architecture has been described as having two components viz., text planning component and surface realization component as shown in Figure 1.1.
As seen in Figure 1.1, the text planning component uses a knowledge base, discourse goals user model, plan library and discourse history to organize the components of the text to form a discourse plan. This discourse plan goes into the surface realization component. The surface realization uses the lexicon and grammar to produce the text. Reiter has segmented the text-planning component into two distinct components viz. content determination and text-planning component. This three-staged pipelined architecture has been regarded as the architecture of any NLG architecture. Accordingly, the important phases of natural language generation can be described as content determination, planning and realization (David McDonald 1988, Denis Carcagno and Lindija Iordanskaja 1993, Appelt 1982).

Content determination deals with determining the various types of knowledge required for the generation of text (Dale and Nicholas Haddock 1991). This knowledge can vary from the level of commonsense knowledge to domain specific knowledge. The next phase, planning deals with the organization of semantic components as well as syntactic components. The final step, realization involves the linguistic mapping of the organized components using syntactic and lexical rules. Reiter’s architecture defines the components based on the functions performed from the computational viewpoint. Another perspective of defining the architecture is from the linguistic viewpoint. From this perspective, generation can be viewed as a set of linguistic functions to be performed to generate a document. Lynne Cahill’s seven-component classification of the NLG process is primarily based on the linguistic viewpoint. The seven component tasks described by Cahill are as follows:
• **Lexicalisation (Lex)**

Lexicalisation means the choice of content words which will appear in the final output text. However there is a distinction between "lexicalisation" and "lexical choice". The first refers to conversion of something typically conceptual to lexical items, while the second refers to deciding between lexical alternatives.

• **Aggregation (Agg)**

Any process has putting together more than one piece of information, which at some other level are separate.

• **Rhetorical structuring (Rhet)**

Determining the rhetorical relations between pieces of information. It involves concepts such as "elaboration", "contrast" etc., and determines how the pieces of information should be related by the text structure.

• **Referring expressions (Ref)**

Deciding how to refer to concepts or entities. This relates to lexicalisation, but it may also be done at a higher level, determining, for instance, that a pronoun is appropriate in a particular case, without determining which pronoun is to be used.
- **Ordering (Ord)**

  The linear ordering of pieces of information may be determined at a fairly high level, but equally it is possible for ordering of sentences in the surface text to be determined at a later stage in the generation process.

- **Segmentation (Seg)**

  Segmentation involves the dividing up of information/text into sentences and paragraphs, and so is the complement of aggregation. The definition of segmentation is not restricted to segmentation into actual text units, but also the segmentation into “sentence sized chunks” of information.

- **Coherence (Coh)**

  Coherence is a cover term for phenomena such as centering, salience and theme processing. Such processing is actually relatively rare in text systems.

  The following table 1.1 shows the correspondences between various component tasks as described by Cahill and the traditional Reiter’s three-staged pipelined architecture. As the table shows, lexicalisation is mostly determined at the sentence planning/surface realization phase, but however, in systems where the input is directly obtained from the user, lexicalisation choice is done at the content determination phase itself. Aggregation on the other hand is mostly confined to the sentence-planning phase except in the systems that get input from the database or knowledge base where aggregation is performed at the content determination phase. Rhetorical structuring is in most cases performed at the content determination phase. Determination of referring expressions can be classified into
determination of the general type of expressions and the specification of the actual surface expressions to be used. Hence depending upon the type of determination performed, referring expressions generation are done either at the content determination or at the sentence planning or at the surface realization phase. Ordering can be at the inter-sentential and intra-sentential level. Inter-sentential ordering is determined at the content determination or sentence-planning phase while intra-sentential ordering is generally performed at the sentence realization level. Segmentation is normally performed at the sentence planning level. Most generation systems do not include the centering/salience component.

Table 1.1 Correspondences between various component tasks

<table>
<thead>
<tr>
<th>NLG Systems</th>
<th>Lex</th>
<th>Agg</th>
<th>Rhet</th>
<th>Ref</th>
<th>Ord</th>
<th>Seg</th>
<th>Coh</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIST</td>
<td>CD/SR</td>
<td>CD/SP</td>
<td>CD</td>
<td>SP/SR</td>
<td>CD/SP</td>
<td>SP</td>
<td>X</td>
</tr>
<tr>
<td>Drafter</td>
<td>CD/SR</td>
<td>CD/SP</td>
<td>CD</td>
<td>SP/SR</td>
<td>CD/SP</td>
<td>SP</td>
<td>X</td>
</tr>
<tr>
<td>Drafter II</td>
<td>CD/SR</td>
<td>SR</td>
<td>(user)</td>
<td>CD/SP</td>
<td>X</td>
<td>CD/SP/ SR</td>
<td>X</td>
</tr>
<tr>
<td>Patclaim</td>
<td>CD/SR</td>
<td>SP</td>
<td>SP</td>
<td>CD</td>
<td>CD</td>
<td>CD</td>
<td>X</td>
</tr>
<tr>
<td>Joyce</td>
<td>SP</td>
<td>SP</td>
<td>CD</td>
<td>SP</td>
<td>CD</td>
<td>CD</td>
<td>X</td>
</tr>
<tr>
<td>Modex</td>
<td>X</td>
<td>SP</td>
<td>CD</td>
<td>(user)</td>
<td>CD</td>
<td>CD</td>
<td>X</td>
</tr>
<tr>
<td>Exclass</td>
<td>CD/SR</td>
<td>SR</td>
<td>(user)</td>
<td>CD/SP</td>
<td>X</td>
<td>CD/SP/ SR</td>
<td>X</td>
</tr>
<tr>
<td>Healthdoc</td>
<td>SP</td>
<td>SP</td>
<td>CD/SP</td>
<td>(user)</td>
<td>SP</td>
<td>(user)</td>
<td>CD</td>
</tr>
<tr>
<td>Ghostwriter</td>
<td>SP</td>
<td>SP</td>
<td>CD</td>
<td>SP</td>
<td>?</td>
<td>CD</td>
<td>CD/SP</td>
</tr>
<tr>
<td>ANA</td>
<td>SR</td>
<td>SR</td>
<td>CD</td>
<td>SR</td>
<td>CD</td>
<td>SR</td>
<td>?</td>
</tr>
<tr>
<td>PlanDoc</td>
<td>SP/SR</td>
<td>CD/SP</td>
<td>CD</td>
<td>CD</td>
<td>SR</td>
<td>CD</td>
<td>CD</td>
</tr>
<tr>
<td>FOG</td>
<td>SP/SR</td>
<td>?</td>
<td>CD</td>
<td>SR</td>
<td>CD</td>
<td>CD</td>
<td>CD/SP</td>
</tr>
<tr>
<td>GOSSIP</td>
<td>SP/SR</td>
<td>?</td>
<td>CD</td>
<td>SR</td>
<td>CD</td>
<td>CD</td>
<td>CD/SP/ SR</td>
</tr>
<tr>
<td>LFS</td>
<td>SP/SR</td>
<td>CD</td>
<td>CD</td>
<td>CD</td>
<td>CD</td>
<td>CD</td>
<td>CD/SP/ SR</td>
</tr>
<tr>
<td>CGS</td>
<td>SR</td>
<td>CD</td>
<td>CD</td>
<td>SP</td>
<td>CD</td>
<td>CD</td>
<td>CD/SP/ SR</td>
</tr>
<tr>
<td>PostGraphe</td>
<td>CD</td>
<td>CD</td>
<td>CD</td>
<td>SR</td>
<td>CD</td>
<td>CD</td>
<td>?</td>
</tr>
<tr>
<td>Komet</td>
<td>SR</td>
<td>SR</td>
<td>CD/SP</td>
<td>CD/SP</td>
<td>CD/SP</td>
<td>CD/SP</td>
<td>CD/SP</td>
</tr>
<tr>
<td>AlethGen</td>
<td>SR/SR</td>
<td>SP</td>
<td>SP</td>
<td>SP</td>
<td>CD</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Proverb</td>
<td>SP</td>
<td>SP</td>
<td>CD</td>
<td>SP</td>
<td>CD/SP</td>
<td>CD</td>
<td>X</td>
</tr>
</tbody>
</table>
1.2 ISSUES IN NATURAL LANGUAGE GENERATION

Language generation is a complex task that requires a considerable amount of knowledge, including domain knowledge and linguistic knowledge at various levels: pragmatics, semantics, syntax, morphology, phonology, etc. Different kinds of knowledge are needed for different subtasks, each kind of knowledge preferably handled by its own data structures and processing mechanisms, generation does not seem to be a homogeneous process. Discourse organization requires pragmatic knowledge, lexical selection requires semantic knowledge, syntactic plans require knowledge of grammar rules, inflection requires morphological knowledge, etc. Given this heterogeneity, the generation process is, at first sight, best viewed as a number of modules reflecting the various strata of knowledge as shown in Figure 1.2.

![Figure 1.2 Various strata of knowledge](image)

Generation needs to address the following aspects of representation and processing.
• **Processing aspects**

How is the generation process decomposed into subtasks, modules, stages or levels? What is the control structure that coordinates the various modules and directs the information flow between them? How can we manage to be efficient while remaining flexible to handle complex demands?

• **Representation aspects**

What information is relevant at the various levels to achieve linguistic expressiveness? What are the intermediate representations that serve as input and output at each level? What are the most adequate formalisms to represent and manage the various kinds of knowledge involved in generation? The process of generation at a very high level of abstraction can be decomposed into two stages, which are realized as the main modules in a generation system. One module determines the content of an utterance, or what to say. The other one realizes its expression in agreement with the rules of a given language: it determines how to say it as shown in table 1.2.

**Table 1.2 Modules of the generation process**

<table>
<thead>
<tr>
<th>What to say</th>
<th>How to say</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>Tactics</td>
<td>(Thompson, 1977; McKeown, 1985)</td>
</tr>
<tr>
<td>Speaker component</td>
<td>Linguistic component</td>
<td>(McDonald, 1983)</td>
</tr>
<tr>
<td>Goal identification and utterance planning</td>
<td>Realization</td>
<td>(McDonald, 1987)</td>
</tr>
<tr>
<td>Deep generation</td>
<td>Surface generation</td>
<td>(McKeown Swartout, 1988)</td>
</tr>
</tbody>
</table>
Different developers use different terminologies to express the two aspects. The first column in the table 'what to say' or strategic component is generally a non-linguistic module, which composes content of a discourse by selecting and organizing information, taking into account communicative requirements of the generation. The second column 'how to say' or tactic component is concerned with the syntactic, lexical, morphological and phonological and is dependent on the language of generation. The generation process can thus be also viewed as having two distinct parts viz-strategic component and tactic component. The Conceptualizer (Strategic component) is a non-linguistic module, which composes the content of a discourse by selecting and organizing information. In doing so, the strategic component must take into account communicative requirements such as those relating to coherence, reference and focusing. The Formulator (tactic component) casts a message into a linguistic form by using the resources of a particular language, i.e. its lexicon and grammar; sentences are built by selecting appropriate words, which are put in the right order and shaped according to morphological and phonological rules. In addition to the strategic component and tactic component, peripheral modules are required for the realization of speech or written text.

The modular approach is useful both from a psychological and from an engineering point of view. From a psychological point of view, there is evidence for autonomous modules: it seems that in human language generation, each module's mode of operation is minimally affected by the others (Levelt, 1989; Fodor, 1983). It has also been argued that the speed of human language generation requires an efficient system where different modules operate simultaneously (in parallel) on different pieces of the utterance. From an engineering viewpoint, decomposition into modules makes the task manageable and testable: one can concentrate on the processing aspects of a specific module.
without worrying too much about the internals of the others, provided that the interface between the modules is well defined. It remains to be seen, however, to what extent it is realistic to have the different modules operate independently, and how they communicate. Various authors deal with the questions of decomposition and interaction differently. One extreme is a sequential architecture with a one-way information flow, where modules are maximally independent. The other extreme is an integrated architecture, where knowledge at all levels acts together. In between the extremes, there are architectures with varying kinds of interaction between the modules (Giovanni Adorni, Michael zock 1996).

Sequential architectures incorporate a one-way flow of information through processing stages ordered in time. Roughly speaking, the strategic component first determines the content to be uttered. It delivers this message to the tactic component, which subsequently attempts to compute an appropriate linguistic form. In a strictly sequential architecture, there is no direct feedback between these stages. The systems Mumble (McDonald, 1983), Text (McKeown 1982, 1985), Naos (Novak 1987a, 1987b), and Wisber (Horacek 1990) are representatives of a sequential architecture. Though the systems described above are essentially modular, they are dictated by sequentiality in time. These systems have the advantage of simplicity.

However, a problem with a one-way flow of information between modules is the need to make early commitments in the construction of an intermediate representation for the next module. The local decisions taken in each module do not necessarily sum up to a good overall result. In particular, this architecture faces the generation gap problem (Meteer 1990): it cannot always be guaranteed that there are words covering exactly the content to be
expressed, nor that there are suitable syntactic patterns that fit a conceptual structure. An avoidance strategy adopted by some systems is allowing the strategic component to 'think' only in terms of the words in the language, thereby trivializing the problem of lexical choice to what Marcus calls 'capital letter semantics'. Systems designed to overcome some of the disadvantages of sequential systems are integrated systems where knowledge at all levels are combined together. An integrated architecture allows different kinds of knowledge to be represented and processed in a uniform way, which is advantageous because of the systematic nature in building the knowledge sources. Kamp (Appelt 1985) is the example of an integrated system. All decisions are taken within a single, hierarchically structured process, so that generation takes place in a continuum of goal-driven and constraint-based processing. However integrated systems have the disadvantage that the different knowledge sources are tightly interwoven and cannot be localized. This hinders maintenance, transparency and the construction of an efficient large-scale system. In between sequential systems and completely integrated systems are systems with varying kinds of interaction between the modules. Interactive architectures involve feedback between different stages of processing at certain points, in particular from the tactic component back to the strategic component, as in Pauline (Hovy 1990) and Popel (Neumann and Finkler 1990, Reithinger 1992). If the different modules in an interactive architecture are well coordinated, they can operate in parallel. This is the case in POPEL, where the different modules operate simultaneously (inter component parallelism). Both modules also work on different pieces of the utterances at a time, thereby realizing intra component parallelism.

Interaction can also be realized in other ways. In Rubinoff's (1992) Igen system, the tactic component provides feedback to the strategic component.
in the form of annotations that tell it how much of the content can be covered by a particular word choice. With these annotations, the strategic component can then determine which choice satisfies its secondary goals.

Another form of interaction is provided by blackboard architectures for generation, as in the Diogenes system (Nirenburg 1988, Nirenburg, Lesser Nyberg 1989). In a blackboard architecture, modules provide information without needing to know exactly which other modules use it. Each module looks on a blackboard for the information it needs and writes its own output on the very same blackboard. An advantage of this architecture lies in the identification of a set of knowledge sources (this is the term for modules in Diogenes), each of which is equipped with clearly delineated competences, thereby strengthening the system's maintainability and providing evidence about its coverage. In addition, blackboard systems allow for flexible processing: the difference in the temporal execution of the various modules stems from the different orders in which triggering information become available. However, the interplay between the knowledge sources on the blackboard needs to be supported by some mechanisms to increase efficiency and to resolve conflicts. In particular, extra control knowledge is needed to assign priorities to knowledge sources, as well as a limited backtracking coupled with simple truth maintenance system.

In order to benefit from the simplicity of a modular architecture while compensating for its limitations, several revision-based approaches have been proposed. These presuppose a limited form of feedback, mediated by monitoring modules that inspect intermediate structures. In the systems KDS (Mann 1988) and Yh (Gabriel 1986) these intermediate representations are amended, whereas in Weiver (Inui, Tokunaga Tanaka 1992) early decisions
which led to unsatisfactory results are undone. A text specific approach using revision is proposed by Robin (1993) in a system that describes sports events. The architecture used in this system is based on a two-pass process. Basically, new information is first generated in a draft, which is subsequently revised to incorporate historical background information. An advantage of this revision-based approach is that it allows rating of communicative goals according to their relative importance.

Having surveyed the major paradigms, the task is to choose the best architecture. A preliminary answer to this question is that there is no universal solution, but the choice depends on the functionality and coverage that the generator is required to have. Stratification architectures with a sequential information flow are widely used and seem sufficient when the task does not require special capabilities. In contrast, special purposes or genres suggest the use of more complex interactions between modules. Robin's revision-based system seems suitable in cases where the type dictates the global organization of the text, further illustrative details being added wherever they fit in. Another example with special constraints is incremental generation, where the demands of real-time uttering require a close coordination between the strategic component and the tactic component. It is this constraint which has led some authors to adopt an interactive architecture (Finkler Neumann, 1989; Neumann Finkler 1990). Another problem to be considered is the cost or effort it takes to build a system. As a matter of fact, one can afford to build a more ambitious architecture if the domain is restricted, because then the complexity of choices will not be as high.
1.3 PHASES OF NATURAL LANGUAGE GENERATION

As already described, the three main phases of NLG are content determination, planning and realization.

1.3.1 Content Determination

Content determination deals with the identification, extraction and representation of information that is to be used for the generation. Content determination concerns with what information to include achieving the particular communicative goal. Identifying input for a domain is a crucial job. The role of world knowledge and commonsense knowledge along with the domain knowledge enhances generation of natural language text. Knowledge of a task guides the choice of what information to include. Content determination requires huge amount of domain dependent data. In this thesis, we select an application, which is a marginally close-ended domain, which restricts the inputs needed for generating a natural language text effectively for achieving the communicative goal. The entire knowledge sources needed for identifying the input and the input components identified should have uniform representation, by means of which, easy access to information can be provided. Since information is conceptual, the information is represented using frames, a knowledge representation scheme which possesses good inference capability. Knowledge sources include domain knowledge, world knowledge and commonsense knowledge. To generate text without deterioration, using the knowledge sources 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.
1.3.2 Planning

In the physical world a plan is a prescription for sequence of actions, that if followed will change the relations among objects so as to achieve the desired goal. One way to represent a plan is by way of a sequence of assertion additions and deletions that reflect physical moments. Plan creation is performed by searching for a sequence of operators that lead from the assertions that describe the initial state of the world to assertions that describe the goal. To perform planning, the various actions have to be captured. A planner starts with the overall task, which must then be reduced to primitive actions. Plan typically list further sub goals, which are expanded. Planning then continues until primitive acts are satisfied. The result is a planner represented as a tree structure in which the nodes represent the goals at various abstraction levels. The root node is the main goal and the leaves are the primitive realization statements at different levels. When the concept of planning is applied to solve AI problems, generally possible routes to the solution are generated as plans. Plans generated are thus composed of operator schemata, provided to the systems for each domain of application. Operator schemata can further be organized into groups of plans. AI planning involves events and/ or actions. Operator schemata characterize the events. The schemata describe events in terms of three conditions and effects.

1.3.2.1 Types of AI planning

AI planning can be categorized based on the different planning techniques used. Systems like GPS, STRIP, HACKER and INTERPLAN are the earliest planning systems. Basically these employ non-hierarchical planning, which involves finding a sequence of operators (i.e. a plan) such that
when applied in sequence to the initial state, the goal state results. In order to remedy the problems associated with the nonhierarchical planning, NOAH (Nets of Action Hierarchies) and ABSTRIPS (Abstraction-Based STRIPS) planning systems represent a plan as a hierarchy of plans where a higher-level plan is an abstraction of the plans at the lower level. Lower level plans describe the detailed action steps needed to achieve the higher-level goal. Skeleton planners use slightly different methods to reduce the search space. Like hierarchical planners, skeleton planners construct a plan by first constructing the skeleton of the plan and filling in the details with lower level plans. However, a skeleton planner does not construct a plan by generating them from a hierarchy of plans, but it produces one by selecting and instantiating one of the prestored plans. The instantiation process during skeleton planning involves searching through vast amounts of domain specific knowledge until the right operator is found. This itself can be a big problem especially when there are conflicts among the instantiated operators. A typical opportunistic planner constructs plan by alternating through two distinct phases. The first phase is an observation phase where different problem solving components communicate with each other via a blackboard to identify the constraints and operations that need to be performed. The second phase is a decision making phase where an operator is selected from the set of operators. Both hierarchical planners and skeleton planners discussed above as well as all the other planners mentioned up to now can be categorized as deliberative planning systems. In a deliberative planning system, a plan for completing an entire task is constructed prior to the execution of any action. Whereas in reactive planning, the order of execution of actions can be altered due to the changed environment; some situations require immediate attention and rapid action. During the course of execution of actions, the initial specification of the goal may be changed requiring replanning of actions. In distributed multiagent planning, there is no centralized controller.
Individual agents would like to think that other agents are working towards the common goal and interest, however, since no agent has a global view of the activities, there is no way to be sure. While an agent is working towards a goal, it is possible that some other agent is working on a goal, which may be competitive or orthogonal to its goal. Therefore, in order to collaborate the efforts of distributed agents towards a converging goal, it is necessary to exchange information among the agents.

1.3.2.2 Elements of Planning in Natural Language Generation

In the previous section we reviewed the techniques of planning in AI, and in the following section the usage of AI planning techniques in natural language generation systems are considered. In order to view the aspects of planning in natural language generation, enumerate the kinds of tasks in natural language generation systems are first enumerated. Almost 20 years ago, when research in natural language generation started, the goal of research was as primitive as to generate a random sequence of sentences that are grammatically correct. For example, (Yngve 62) describes a system that uses generative context free grammar and a random-number generator to generate grammatical sentences. Later on, efforts were made to achieve a more general goal of building a component of machine translation paraphrasing system that takes some internal representation of the "meaning" of a text and converts it into a grammatically correct realization of the text. Thus, the focus of these earlier natural language generation research was on the mechanics of constructing phrases and sentences using linguistic resources, and the task of natural language generation was more or less clearly defined as converting the pre selected set of contents into a unit of text that conveys the intended meaning in the given representation of the contents. However, the definition of the natural language generation tasks
became less clear as the representation of the content and the level of details in specification of the content varied widely from one application to another. The process of content formation is called deep generation and that of transforming the content into text is called surface generation, and the part of the generation system that is responsible for the task of deep generation is called strategic component and that for the surface generation is called tactical component. Appelt also acknowledged the task of both deep generation and surface generation as part of natural language generation task. Instead, he adopted the work of Cohen and Perrault on speech acts and implemented a system called KAMP (Knowledge And Modalities Planner) in which an integrated planning module plans the content of the output sentence, and realizes it in short sentences which are served as illocutionary acts to influence the hearer to perform an action. However, while building any natural language generation system, organization of the content, plays an important role. While organizing the content of any document or text, various types of knowledge such as domain knowledge, social goal knowledge etc have to be incorporated. Moreover, planning for natural language generation involves the knowledge of the user at various levels. Hence, planning for natural language generation is different from AI planning. In NLG planning the initial state are the specified communicative goals, while the final state is a cohesive linguistically correct text that satisfies the specified goals. In general, planner for natural language text emphasizes on ‘how to say’ component of natural language generation.

1.3.3 Realization

The next step is realization, which involves mapping the structured component obtained by content determination and text planning process with the linguistic component. A realization specification is a composite structure
that is brought together for a specific situation. The output of realization will be a choice among pre-determined set of alternatives. When a choice is made a well-formed structure is constituted to meet the schematic description. The realization component depends on sophisticated knowledge of the language’s grammar and rules of domain, which typically constructs a syntactic description of the text. It emphasis not only on linguistic component, that also on the knowledge of the criteria that dictate how the forms are used. In general, natural language generation posses the three phases viz content determination, planning and realization. However the emphasis and methodologies to be adopted by each phase can be determined by considering the various applications of natural language generation listed in the next section.

1.4 APPLICATIONS OF NLG

The various applications of NLG can be broadly classified under automatic text generation and automatic text summarization.

Automatic text generation posses the following types of applications:

- Validation of formal specifications (requirements engineering)
- Automatic technical documentation generation
- Automatic weather reports from raw data
- Explanations in expert systems
- Medical informatics
- Machine translation translation between natural languages
- Translation to multiple natural languages from one source representation
Automatic text summarization can be used for:

- Summarizing newspaper text (for journalist, business intelligence, technology intelligence, etc)
- Summarizing reports (for parliament members, investigators, businessmen, etc)
- Search engines to extract keyword and to obtain summaries of the found text
- Search in foreign languages and obtain an automatic summary of the machine translated text
- Extracting keyword and summaries of email for SMS in mobile phones
- Summarizing text which has been downloaded from the Internet from a WAP mobile phone
- Letting a computer read summarized www pages on a mobile phone.

In this thesis, automatic text generation in general, and automatic software document generation in particular have been considered as the application because while automating software documents, productivity and the quality of software is increased.

1.5 ISSUES IN SOFTWARE DOCUMENT GENERATION

Software document generation is a challenging task. For different types of activities performed across the software development life cycle, different types of documents have to be prepared. While, manually preparing software documents, sometimes, due to lack of data/information acquisition,
documentation may be incomplete and inaccurate. The documents generated during a software development project serve various purposes (Williams 1988):

- Documents facilitate communication between individual and teams working on different parts of the project.
- Documents provide the tangible evidence of progress that project managers require
- Documents become part of the archival history of the project.

Software documents vary widely in their formation, readability and precision. The extent to which a document can be processed by a computer depends on the extent to which the document is formalized (Watts S.Humphrey 1999) (Weiss Edmund 1991, 1995).

In order to prepare a complete and perfect document, role of various types of knowledge, such as, domain knowledge, world knowledge, linguistic knowledge etc. is required. In general, software engineering is knowledge intensive. Software projects make extensive use of documents. Indeed, the waterfall model of software development has been described as document driven (Ghezzi et al 1991). Various types of documents have to be prepared based on status of the user. In order to produce a good document, there is a need for complete coordination between software designers, developers and documentalists. This is often very difficult to achieve in an industry scenario. Moreover, separate documentalists do not exist in most organizations and the job is often left to either consultants who do not have a comprehensive view of the software, or to developers themselves who do not posses the necessary skills to produce good software documents. In order to develop uniform and consistent software documents, we have to incorporate standards while
preparing documents. There are a number of software document standards available. And depending upon the customer for whom the product is designed, the document has to follow the appropriate standards. This adds to the complexity of manual software documentation.

Moreover there will be change in the documents due to the evolution of software. There is a need to maintain software document to reflect these changes. The software documents have to be constantly maintained while at the same time adhering to standards. The availability of good documents need not be emphasized since this is often as important criteria by which actual users of the software evaluate it. Hence there is an urgent need to automate the process of software documentation. In this thesis, an attempt has been made to automatically produce software documents from software requirement specification techniques.

1.6 MOTIVATION AND CHALLENGES IN AUTOMATIC SOFTWARE DOCUMENT GENERATION

Software documentation is a time consuming task and often production of a good document is one of the bottlenecks in the software development life cycle. The automatic software documentation addresses many of the issues in manual software documentation and also provides a challenging task for natural language generation. Automatic software documentation avoids the necessity of having complete coordination between developers, designers and documentalists by automatically extracting knowledge about software artifacts from the software requirement specification techniques. Automatic software documentation addresses the issues of producing documents to meet
different software document standards and allow tailoring of document to cater to different user levels.

Automatic software documentation is a fairly restricted domain and hence allows an effective content determination in the NLG life cycle. Though the SRS techniques provide a deterministic input to the content determination phase, there is a need to extract inputs 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 relationships diagrams, state transition diagrams and control flow diagrams. The software artifacts identified as inputs have to be organized and represented using a uniform formalism. Appropriate hierarchical and temporal relation between the entities also has to be part of the representation techniques. There is often a need to make appropriate changes in documentation due to change in application domain or due to evolution of software. Automatic software documentation needs to adapt to such changes in the document. This also means that the design of the knowledge representation should be adaptable to such changes. The organizational structure of the software document needs to be designed based on the software document standards. The planning phase of the automatic software documentation needs to provide mechanism for organization of document according to different standards, but constrained by linguistic constraints. The automatic software documentation described in this thesis attempts to address some of the issues and challenges described above.
1.7 PREVIEW

Figure 1.3 shows the broader perspective of the automatic software document generation system: Natural language generation is a highly complex task whose automation is performed by accessing huge amount of knowledge sources such as domain knowledge, world knowledge and common sense knowledge. Software documentation is chosen as application for our work because of its closed ended domain i.e. even if the domain is not clearly specified the inputs required have to be properly identified for producing corresponding document. As in any other NLG system, the first task is content determination phase. Inputs 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 relationships diagrams, state transition diagrams and control flow diagrams. In the next module, identified inputs for generating a natural language text are organized using knowledge representation technique, frames. In this work, the frame representation scheme has been enhanced with specially designed new component called perspective descriptors, which tries to improve the epistemological status of the frame structure and adapts to changes in the software.

In order to represent the process sequentiality among the software entities, causal links have been introduced in the knowledge representation schema in addition to the hierarchical links that depict the structural organization among them. The next phase is planning. Adaptive planning is
Content Determination

required to deal with producing documents at different levels. For automatic software document generation a generic standard which consists of Aim, Introduction, Purpose, Objective, Functional Behavior, Informational Behavior, Procedural details, and all sub process details using document design patterns,
has been used. The document design patterns are designed to satisfy the linguistic constraints also. Generally, patterns provide recurrent solutions. For each and every component of the software standard, specific patterns such as initiator, instantiator, illustrator, comparator, counter, fetcher etc. that matches with the linguistic constraints have been designed. The output of the planner is passed to the realization phase to obtain the complete and perfect document.

Realization, maps the identified content from the content determination phase organized using planner phase, and generates actual text which meets syntactic and semantic constraints. The realization tasks are: structure realization and linguistic realization. Structure realization deals with choosing markup to convey document structure and linguistic realization deals with insertion of function words, choosing correct inflection of content words, order words within a sentence, and applying rules. Grammar has been designed for automatic software documentation, based on the three views viz. functional view, behavioral view and informational view. Using this generative grammar, the document is generated.

The remaining part of the thesis is organized as follows: Chapter 2 deals with survey of NLG systems. Chapter 3 focuses on need for software documentation. Chapter 4 represents content determination, where the inputs are identified for generating automatic software documents and represented using knowledge representation frames extended with perspective descriptors and causal links for increasing reusability and adaptability. Chapter 5 enhances planning for software document generation, which has been designed using specially, designed document design patterns. Chapter 6 gives realization of natural language text using BNF grammar. Chapter 7 provides conclusions of this work and presents possible future enhancements.