CHAPTER 5

PLANNING

5.1 INTRODUCTION

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 (Allen J Hendler and Tate 1990). 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 (Haller 1996). To perform the 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 (Appelt 1985, Srivastava 1998, Tate 1976, Pollack 1997, Wilkins 1998, 1999, Moore et al 1993, Steave Beale 1998). 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 the AI problems, generally are possible route to the solution is generated as plans. Plan generated is 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 conditions and effects. In the first section general AI planning is discussed. In this work the planning of documents is tackled using patterns.

5.2 TYPES OF AI PLANNING

AI planning can be categorized based on the planning techniques as follows:

5.2.1 Nonhierarchica l Planning

Systems like GPS, STRIP, HACKER, and INTERPLAN are the earliest planning systems. Basically this technique involves finding a sequence of operators (i.e. a plan) such that when applied in sequence to the initial state, the goal state results.

5.2.2 Hierarchical Planning

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. In addition to this plan decomposition hierarchy, some systems also represent the actions and objects involved in a plan using subsumption relations in order to achieve the effect of having an even finer degree of granularity among the actions(Minton S.Bresina 1994).
5.2.3 Skeleton Planning

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. A prestored skeleton plan consists of the outline of solutions for many different kinds of problems and it is refined by instantiation of problem solving operators containing information for solving a specific or common problem in detail; these operators are found by the skeleton planner from the pool of domain specific knowledge.

5.2.4 Least commitment Planning

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.

5.2.5 Opportunistic Planning

Opportunistic planning takes an approach that is totally different from the goal driven planning approaches taken by other planning techniques discussed so far. A typical opportunistic planner constructs a plan by alternating through two distinctive 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. The selected operator relaxes the constraints or contributes to the operations identified by the observatory phase. After an operator is selected, the observation phase takes over and each of the problem solving components reevaluates the situation to post another constrained operation. This two phase 1 Actually, he names them strategy space, design space, and instantiation space. 2 process continues until there are no more operations that need to be performed. The sequence of decisions made through this process is a plan that achieves the goal of the planner. A plan constructed by the opportunistic planner may follow an orderly path and produce a neat top-down expansion. However, it is possible that during the planning process, some of the initial sequence of actions may have imposed limitations on the latter actions, and thus it is possible that a less than ideal plan is generated.

5.2.6 Reactive planning

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. However, for certain tasks, planning and execution of a plan must be interleaved because of the following characteristics of the real life planning problems: A planning system with limited resources must decide when to start planning, when to stop planning, and when to act; if a rock is falling over one's head, he would rather take any direction to get out of the way than stand there until he finishes planning for the next move he will make once he gets out of the way. The state of the world after execution of an action necessitates new goals or eliminates execution of already planned actions. 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.

5.2.7 Increasingly Reactive Planner

As demonstrated above, reactive planners have the advantage that they can behave robustly in dynamically changing world without having to model the worlds in advance. Also they are extremely responsible because they do not need to do the combinatorial explosive search that slows down other search based deliberative planners. However, pure reactive planners have limitations also. Since they do not maintain any model of the world or explicit goal structures, they cannot be used for solving tasks that require intensive domain knowledge. That is the reason why although pure reactive systems are often used for real time tasks like driving and walking, they are not suited for playing expert chess. For this reason, the increasingly reactive planner is a deliberative planner which incorporates both a reactive component and a search based planning component such that an agent in his planner, called Theo Agent, reacts when it can, plans when it must, and learns using an explanation based learning mechanism by augmenting its reactive component with newly formulated stimulus response rules whenever it is forced to plan. Therefore, what starts out as a deliberative planner with a minimum reactive component becomes increasingly reactive as its learning component acquires new stimulus response rules that eliminate the need for planning in similar subsequent situations (Veloso M and Stone P 1995).
5.2.8 Cooperative Distributed Planning

For a large task with more than one sentient agent collaborating together towards the completion of the task, it is necessary to coordinate the efforts of the agents in order to achieve the task while taking advantage of distributed information, resources, and expertise. With the emergence of the new field in AI called parallel and distributed AI much research has been conducted to solve the problem of coordinating the actions of the individual agents in order for them to work together effectively. The variety of approaches can be grouped into the following three categories based on the nature of the distribution of the agents: 1. Centralized Multiagent Planning: This is the least distributed kind. A single agent (master agent) makes a plan and distributes pieces of the plan to other agents (slave agents) that carry out the task and report back to the master agent. Since this kind of planner is built to avoid inconsistent conflicting actions, it is typically used in a domain where unexpected conflicts for a resource can be costly.

5.2.9 Distributed Multiagent Planning:

This is the most distributed planning situation of the three discussed here. 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 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. One natural solution is to allow agents
to communicate with other agents so that an agent can gather enough information to create a plan. By assuming that all the agents are rational and thus building plans that contribute to the solution of the problem, eventually the effort converges to the solution. However, sometimes direct communication between two arbitrary agents is not possible (Wickler et al 1997).

5.3 PLANNERS

Various planners were designed for various purposes based on the domain. For any type of application, the planner should decide the plan structure, search space and search control techniques (Manuela Veloso et al 1994). The plan structure deals with how the content is organized. Using action ordering as plan structure various systems such as Greens Systems, Waldinger system, SIPE (System For Interactive Planning And Execution which generate plans and uses parallel plans) STRIPS.

All the above systems use partial plan and deal with search space describing world states. HACKER is a learning program, which also use partial plans.

The various search control techniques used are BFS and means ends analysis etc. Kowalski’s System used BFS on the length of the plan (Power 2000, 2001). STRIPS used means end analysis and BFS. The other planners available are WARPLAN, which uses actions regression to constructs plans. NOAH used procedural nets. DEVISER has time windows, duration for actions and goals. PLANX uses plan recognition /common sense planning. CAPLAN is a domain dependent planner but ensures a systematic search through the search space. GRAPH PLAN uses STRIPS domains and the flow of true values
through a graph. RAP plans primitive robot actions through well-defined goals. SIMS is an intelligent system that has an intermediate layer that acts as a mediator between information sources and human users or application program. UMCP universal method composition planner is a hierarchical network planning system that performs automatic and interactive search and has effective GUI. XFRM is a planner for robotic application. Oplan (open planning architecture) is a planner (Ken Currie and Austin Date 1990) used for project management assembly, integration & verification. TRIPS is a collaborative planning assistant, TRAINS a planner used for Natural spoken dialogue and interactive dialogue, SGP is sensory graph plan. Peba II is a text planning system where discourse schemes specify structure and content such as compare & contrast, compare properties and description. Deep directed text planning is also another planner used for text planning. AO plan representation is organized around the notion of goal reduction and prodigy architectures are used for learning based on planning. Auto Brief is an experimental system that automatically creates interactive presentations in coordinated text and information graphics.

5.4 ELEMENTS OF PLANNING IN NATURAL LANGUAGE GENERATION

In the previous section the techniques of planning in AI are reviewed and in the following section we will look at the usage of AI planning techniques in natural language generation systems. The purpose of this section is to bridge the two sections by reviewing the field of natural language generation from the perspective of the AI planning and identify the elements of planning in natural language generation in AI planning terms. In order to view the aspects of planning in natural language generation, first enumerate the

Planning for the text generation differs from the general AI planning. While general AI planning deals with states and state dictated application of operators, in text planning, the application of operators is dictated by content organizational constraints. Software documentation deals with close-ended domains. Adaptive planning is required to deal with producing documents at different levels. 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. 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.

During the hierarchical planning, operators are primitive or non-primitive, non-primitive operators contain decomposition plans, plan must be consistent, no ordering or variable inconsistencies, every effect of operator must be asserted by at least one step of plan, every precondition of the steps in plan must be asserted by a step in the plan or be a precondition of the operator, hierarchical plans allow a way to store previously computed plans, loops until solution, each time selecting a sub goal and a decomposition. In this work hierarchical planning is preferred for NLG because if it is required to generate documents at various levels depending on status of user, it can be generated.
5.5 HIERARCHICAL PLANNING

Hierarchical planning can be better than non-hierarchical planning if the following properties hold:

- downward solution property, can prune other abstract solutions
- Upward solution property: if an abstract plan is inconsistent, then there is no primitive solution of which it is an abstraction
- Can prune all descendants of any inconsistent abstract plan
- Decomposition and sharing
- Two decompositions of two steps of an abstract plan may need to share steps
- When adding operators, consider sharing as an alternative choice point
- Or, go ahead and merge operators, but let a critic detect the need to share
- Decomposition vs approximation
- An approximation hierarchy in which preconditions are assigned a criticality level can represent an operator
- Planning can occur at a high criticality level (low value) first, then at less critical levels
- Motivation: solve most highly constrained preconditions first

More Expressive Operator Descriptions
- Conditional effects
- Effects now look like "e [when c]"
- Consider c as a sub goal only when e is protected by a causal link
Given a causal link $S_i \rightarrow c \rightarrow S_j$ and an effect ($(\neg c')$ when $p$), ensure $p$ does not hold when $c$ unifies with $c'$ - This is called confrontation.

- Negated and disjunctive goals:
  
  Prove $(\neg p)$ using an operator with $(\neg p)$ effect, or $p$ not true in initial state

- Backtracking handles which of $(p$ or $q)$ is used in a precondition

- Could split operator into two, put forces commitments

While performing hierarchical planning in order to obtain the content in a standard form patterns (Uma G.V, Geetha T.V 2001C) are used.

5.6 PATTERNS

Patterns have shown to be an effective means of capturing and communicating software design experience. A pattern can also be used to create instances of it. A pattern is a form that appears in a context (Martin Fowler 1996, Martin Riehle 1997, Coad P 1992, 1995, Richard Gabriel 1996, Vlissides, Coplien et al 1996). Both form and content of a pattern are abstraction desired from our concrete experiences. The form of a pattern is finite, but the form of its instances needs not to be finite. The context is potentially infinite. The form describing a pattern consists of a finite number of components and relationships. During the software design a conceptual model of the application domain is needed that evolves with our system. Such an application domain should be formal. Usually, it consists of a set of related descriptions based on the concepts and terms of the application domain, comprising different view points of the various groups involved in software
development process. A conceptual pattern is a pattern whose form is described by means of the terms and concepts from an application domain. Conceptual patterns have been originally called Interpretation and high-level design patterns (Riehle 1995). These conceptual patterns should be geared towards a restricted application domain. While considering the activities related to technical design of a system, a model is considered which relates to the conceptual models of the application domain, but takes into account the need for reformulating this conceptual model in terms of the formal restrictions of a software system. This is the formal traditional software design model. It is geared towards software construction. A design pattern is a pattern whose form is described by means of software design constructs, for example, objects, classes, inheritance, aggregation and use-relationship. We can use software design patterns to build and understand a software design model. A design patterns describes the structure and dynamics of its components and classifies their interplay and responsibilities. Design patterns should fit or complement the conceptual space opened by their conceptual patterns. Software design is facilitated considerably, if design patterns can be related to the conceptual patterns used to describe the application domain model that is if they help to realize the conceptual patterns and metaphors on the concrete design level. Apart from using patterns for software design, they can also be used for programming. In constructing the software system, the application domain and the software design model are integrated and result in system implementation. A programming pattern is a pattern whose form is described by means of programming language constructs. We can use these patterns to implement a software design. A district pattern is required in order to arrange and describe sets of related patterns, during software development. Each pattern is always viewed within its context. Hence, pattern can be ordered as nodes of a directed graph. For textual presentation, one can linearize the graph breadth first. Each
pattern/context pair should be proceeded by all pattern/context pairs that are needed to understand it. Conceptual patterns logically precede design patterns, which logically precede programming patterns.

5.6.1 Practical Benefits Of Patterns

1. Design patterns coordinate the entire process and community. They provide a standard vocabulary among developers. They communicate information between designer, programmer, and maintenance programmer at a significantly higher level than individual classes or functions.

2. Design patterns can be used reactively. They can be used as a documentation tool to classify fragments of a design, making it easier for a team to absorb new developers.

3. Design patterns can be used proactively. They are more than a document tool. In addition, they can be used to build robust designs with design level parts.

4. Design patterns can be used to give the software a hinge. It has hinges or hotspots in the specific designated places where the software is developed to adapt.

5. Design patterns can turn a tradeoff into a win-coin situation. They provide a quality of recursive structures.

6. Design patterns constrain, maintenance programmers. Hopes are relatively expensive to design and build.

7. Design patterns let management reward self-directed designers.

Software design patterns describe proven solutions to recurring software design problems. Knowledge of these patterns increases designers'
abilities, leads to cleaner and more easily maintained software, speeds up implementation and test, and helps programmers document and communicate their designs.

The catalogue collects general-purpose patterns from a variety of sources. It includes older patterns such as Module and Layers as well as modern, object-oriented patterns such as Observer and Visitor. A software design pattern describes a family of solutions to a software design problem. It consists of one or several software design elements (such as interfaces, classes, objects, methods, functions, processes, threads, etc.), relationships among the elements (for example association, inheritance, delegation, invocation, and creation), and a behavioral description. The purpose of design patterns is to capture design know-how and make it reusable. Design patterns can improve the structure of software, speed up implementation, simplify maintenance, and help avoid architectural drift. Design patterns also improve communication among software developers and can empower less experienced developers to produce high-quality designs.

For automatic software document generation we have framed our own standards, which consist of Aim, Introduction, Purpose, Objective, Functional Behavior, Informational Behavior, Procedural details, and all sub process details using document design patterns. Patterns provide recurrent solutions. For each and every component we have designed our own patterns such as initiator, instantiator, counter, fetcher etc. that matches with the linguistic constraints. The output of the planner is organized as descriptive statements, which is obtained at the realization phase to obtain the complete and perfect document.
A good pattern solves a problem. Patterns capture solutions, not just abstract principles or strategies. Patterns capture solutions with a track record, not theories or speculation where the solutions are not obvious: Many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly—a necessary approach for the difficult problems of design. It describes a relationship. Patterns don't just describe modules, but describe deeper system structures and mechanisms (Stephen Busemann 1991). A pattern language defines a collection of patterns and the rules to combine them into an architectural style. Pattern language describes software frameworks or families of related systems. Alexander describes pattern as follows: Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces that occurs repeatedly in that context, and a certain spatial configuration, which allows these forces to resolve themselves. As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant. Each pattern is a three-part rule, which expresses a relation between a certain contexts, a certain system of forces, which occurs repeatedly in that context, and a certain software configuration, which allows these forces to resolve themselves.

Pattern elements are name, context, problem, forces and solution:
Name - A meaningful "conceptual handle" for discussion. Context defines how the problem occurs / when the solution works. Problem deals with statement of the problem / intent of the solution. Forces explain the Trade-off, goals + constraints, motivating factors/concerns. Solution tells how to generate
the solution. For example if we consider software development we have the following software patterns:

Design Patterns (software design; often object-oriented): architecture (systems design), design (component interactions), programming idioms (language-specific techniques/style), Analysis Patterns (recurring & reusable Analysis models), Organization Patterns (structure of organizations/projects), Process Patterns (software process design), and Domain-specific patterns. Software Patterns help us because they solve "real world" problems, rationale, reuse wisdom and experience of master practitioners, convey expert insight to novices, form a shared vocabulary for problem-solving discussion, show more than just the solution: context (when and where), forces (trade-off alternatives, misfits, goals + constraints), resolution (how and why the solution balances the forces). Software Patterns are not restricted to software development phases. The patterns can be designed for any real time application where reusability is required. Software Patterns are recurring solutions to common problems of design, Practical/concrete solutions to real-world problems, Context specific "Best-fits" for the given set of concerns/trade-off, a literary form for documenting best practices, a shared vocabulary for problem-solving discussions, an effective means of (re) using, sharing, and building upon existing wisdom/experience/expertise. The figure shows the hierarchical decomposition of processes in the Routing management system. Patterns such as initiator, will initiate the process to hierarchically subdivide if there exists process/processes. Instantiator pattern will instantiate sub problem tester etc., counter pattern delivers the total number of processes present. Fetcher pattern fetches inputs required for executing the various sub processes from the main process.
5.7 PLANNER FOR AUTOMATIC SOFTWARE DOCUMENT GENERATION

The following Figure 5.1 shows the planner for automatic software document generation. The plan generated by the planner is a complete plan where all the assertions and goals are satisfied. The various components are as follows: Plan manager, planning decisions, plan representation. The plan manager consists of plan construction that constructs new plans; plan generation, which generates from the existing pattern. A pattern is a description of a recurring pattern of architectural elements and a rule for how and when to create those patterns. (Ontologically, the attribute generative refers to a pattern’s content and epistemologically descriptive refers to pattern’s form). Plan recognition is used to recognize the identified inputs easily depending on the need and status of the user. Plan selection, selects the necessary plan that matches with the standards and templates. Plan inference, infers the plan and if there is any need to adaptability accordingly the output of the planner can be changed. The output of the plan manager is passed further to the plan representation component. There are two types of plan representation methods: Action ordering, state space ordering. Based on the operators designed with respect to the domain the planner will characterize the output plan construction and plan inference plays an important role (Uma G.V and Geetha T.V 2001a).

Plan construction

Given a goal state, two major tasks need to be done to produce a plan to achieve that goal.
Figure 5.1 Planner for Automatic software document generation
1. Backward chaining: Given a goal G find an action A, that has G as one of its effects. Then evaluate precondition of A.

2. Planning at different levels of abstraction.

**Plan inference**

This is performed based on the original beliefs, Expectation and alternatives (observed action).

Given an initial world state W & a goal state G a plan is a sequence of actions that transforms W into G. Every utterance is the result of several actions (or) speech acts.

**Operators**

Plan operations are essentially used for achieving effective communication goals. Each operator represents 1. an introduction of the effects that the operator is intended to have a users state, 2. a list of constraints that must hold the operator to be applicable match a goal pattern, if one fails, the system automatically tries another. Backtracking occurs, if necessary to undo the effect of all changes to the database resulting from the body of theorem to the point of its failure. 3. Production Systems: It consists of a rule whose procedure called the body of the rule is matched with the associated pattern. Pattern matches the database and then its body is executed. Apart from these operations, time, properties and events should also be considered (Gamma, Helm, Johnson 1994, Dirk Riehle 1994, Coplien and Schmidff 1995, Busctimann, T. Mernier et al 1996). Representing action has relied on the dynamic logic. These formalisms are convenient for plan recognition. Time is
linear and time intervals are individuals each pair related by one of Allen's interval logic relations: Before, Meets, overlaps and so on. Event tokens are also individuals and event types are represented by unary predicates. Various role functions on event token yield the parameters of the event including its time.

**Action oriented operations**

The various action oriented planning operators that can be used to automate the software documentation are: Specify, Expansion, Identify, Request, Inform Order, Operator, Fit, Search and Choose, Elaborate. These operators have to include the pattern based descriptions (effects, preconditions etc) as well as the numerical constraints and plan fragments outlining the action expansion details of the high level operator. The planner for automatic software documentation should also map with the software standards.

**5.8 INCLUSION OF STANDARDS**

The following standard is designed for automatic software documentation. Standardization is very important in any documentation because of its uniformity and consistency. Based on the status of the user the template that has been designed for various services rendered by software development phases and the status of the user the standard will be instantiated.

In order to produce an effective and standardized document we have to follow the available standards (Uma G.V., Geetha T.V. 2001). The standards will constitute various phrases, patterns and models of sentences. If we have to generate sentences according to the standards we have to use document design patterns. Some of the patterns that are used for writing any document from the linguistic point of view is described as follows: There are ten patterns that are useful in software documentation to generate most appropriate and effective text. Patterns explore, amplify, clarify the given term and also it gives texture and immediacy to the given term. The following list shows the ten patterns.
The patterns that have been identified for automatic software documentation have a one to one relationship between the components of the documentation standard, and the origin of the standard for extraction of necessary contents for producing the software document is shown below (Uma G.V., Geetha T.V. 2001b):

<table>
<thead>
<tr>
<th>Component</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aim</td>
<td>can be obtained from context analysis diagram</td>
</tr>
<tr>
<td>Introduction</td>
<td>first level DFD.</td>
</tr>
<tr>
<td>Functional description</td>
<td>Second level DFD.</td>
</tr>
<tr>
<td>Behavioral description</td>
<td>Entity Relationship diagram</td>
</tr>
<tr>
<td>Informational description</td>
<td>Entity Relationship diagram</td>
</tr>
<tr>
<td>Process details</td>
<td>first level DFD, Second level DFD.</td>
</tr>
<tr>
<td>Glossary</td>
<td>first level DFD, Second level DFD and Entity Relationship diagram</td>
</tr>
<tr>
<td>Interaction matrix</td>
<td>first level DFD, Second level DFD and Entity Relationship diagram</td>
</tr>
</tbody>
</table>
Process description : first level DFD, Second level DFD and Entity Relationship diagram and state transition diagram.

Hierarchy chart : first level DFD, Second level DFD and Entity Relationship diagram and state transition diagram

The contents of the corresponding components of the document standard are obtained from the patterns and the mapping is shown below:

Aim : Definition
Introduction : Definition, Illustration
Functional description : Circumstances and Details
Behavioral description : Process
Informational description : Classification and Division
Process details : Process, Cause and effect
Glossary : Clarification and restatement.
Interaction matrix : Illustration
Process description : Analogy, Comparison and Contrast
Hierarchy chart : Process .

5.9 DOCUMENT DESIGN PATTERNS "PATDOC"

The vaguely specified linguistic patterns of the previous section need to be precisely defined using design pattern methodology. Thus each linguistic pattern is given a definition, solution, the forces that act on them and the context in which they work. In this way each linguist pattern we have identified
is converted into a directly usable document design pattern "PATDOC". Now we proceed to define the different document design patterns.

The description of patterns are shown below:

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem definition</td>
<td>To explore the term, terms are words, data that is stored in frame structure that is obtained from various software requirement specification techniques such as DFD, ERD and STD, that is extracted for the document.</td>
</tr>
<tr>
<td>Solution</td>
<td>It amplifies the term by focusing on its connections with similar things. It specifies the meaning by focusing on the unique traits of the term.</td>
</tr>
<tr>
<td>Forces</td>
<td>Identify the appropriate class for the given term, the term may have more than one class. but selecting the class among many depends upon term's topic and then put the term in the identified class it will limits the term's meaning. Differentiate the term to be defined from other members of the class defines the term's meaning.</td>
</tr>
<tr>
<td>Context</td>
<td>These patterns are useful when the terms are ambiguous, controversial or unfamiliar</td>
</tr>
</tbody>
</table>
Example

Definition

Term : Safe home software
Class : Process

Differentiating with other members : This process is produces software system for home security.

Defined Text

Safehome software is a process used to produce software system for home Security.

This definition pattern is required to define the problem that is used in project plan.

2. Pattern Name : Illustration
Problem definition : To clarify the meaning of the given subject.
Solution : It provides illustrations and examples for the given term.
It provides the meaning and skeleton for the document
Forces : Identify the type of illustrations such as short, quick, vivid and extended
Identify the examples that is specific to the terms meaning.
Use graph or data subject to domain specific examples.

Context: When the document needs some kind of sense(or) some kind of judgement.

Example

To document the weather monitoring system we may have to use graph to illustrate the particular day's weather.

This pattern describes in the entire design procedures such as data design, Interface design, Procedural design and architectural design.

3. Pattern Name: Circumstances and details.
   Problem Definition: To explore the term's circumstances and details.
   Solution: It provides the particular characteristics of the term.
   It identifies the marks that distinguishes a term from all other terms
   Forces: No special patterns for this. It is similar to development by definition or development by illustration.
   Identify the particular aspects about the given term.
   Pile up the details one after another in order to show as much of the texture that is present the details in some kind of order.
Choose the pattern either definition or illustration according to the nature of term

Context: When the document needs more details about the particular subject.

This pattern is used to obtain the environmental constraints and to get the overall structure.

4. Pattern Name: Compare and Contrast

Problem Definition: To explain the term by comparing or contrasting features.

- It provides setting for the term that is to be explained
- It provides backdrop or scene for the term.

Forces: Identify the features of the given term. Most things that can be compared are probably different in more ways than they are alike. Most things that can be contrasted probably share common features. So select features that are worth comparing or contrasting.

Context: When the term to be documented needs connection with the world.

This pattern is used to compare the features in order to check the quality of the developed software.
5. **Pattern Name** : Analogy  
**Problem Definition** : To explore the agreement or resemblance in certain respects between otherwise dissimilar things  
**Solution** : It provides the similarities between two things.  
**Forces** : Identify the features that are similar for two taken things. Define the analogy using reasoning in which resemblances are inferred from others that are known—that is one reasons that if two things are alike in some respects, they must be alike in others.  
**Context** : These patterns are very useful when the documentation needs to document the similar things.  

This pattern is used to estimate resources required for analogous applications.

6. **Pattern Name** : Classification and Division.  
**Problem Definition** : To classify or give the division the term belongs to for the given term.  
**Solution** : It refines the term from general class to specific class.  
**Forces** : Identify the major class for the given term and present an account of how this term shares features with other members of the class, for classification.
For division divide the major class into specific groups and give an account of this specific groups.

Context: This pattern is useful when the term belongs to a larger group and is capable of being divided into smaller segments.

This pattern is used to classify according to the functional procedure in turn can be used for software subcontract management.

7. Pattern Name : Analysis
   Problem Definition : To analyse the given subject.
   Solution : It describes the nature of a subject by separating it into its parts.
   Forces : Identify the sub parts of the term that are having specific purpose, and divide the term into its parts just for the sake of division, the of dividing a subject into its parts should somehow support and fulfill purpose in writing.
   Context : This pattern is useful when the document needs the term to divide into its subparts in a variety of ways, and when it needs justification for continuing to scrutinize the term.
Example

Term : Functional Description
Identified Subparts: eight subparts

Subparts Expansion

Safe home software system
<Use possible patterns to expand this subpart like Definition, Classification etc.>
Interact with user
Monitoring Sensors
Text

<define functional description using Definition pattern like> functional description is description which is used to show the various process, sup process and its request and reply for the selected software system.

Safe home software system contains eight processes
1. Safe home software system
2. Interact with user
3. Monitoring sensors etc

Safehome software system is a contains 6 subprocess such as
1. Interact with user
2. monitoring sensors
3. display information etc..
It gets the request user commands and data from Control panel and sensor information from sensors. Interact with user is a process which contains no sub process. It gets the request from user commands and data from the user.

<table>
<thead>
<tr>
<th>8. Pattern Name</th>
<th>Cause and Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Definition</td>
<td>To explore, amplify and clarify the term through cause and effect.</td>
</tr>
<tr>
<td>Solution</td>
<td>When the term is important as an effect, it explores how the term it came into being. When the term is important as a cause it explores what consequences things have i.e. how they matter hereafter.</td>
</tr>
<tr>
<td>Forces</td>
<td>When the term is important as an effect. Identify the source it came from, the term may have more than one cause. Identify the related cause for the term effected. When the term is important as a cause, identify the effect the related item cause.</td>
</tr>
<tr>
<td>Context</td>
<td>This pattern is useful to show the full significance of a term to show that it matters – it comes from somewhere (it arises from a cause or causes), and it goes somewhere (it has some effect, it carries some weight of meaning).</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Term</th>
<th>Alarm Signal(as an effect)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sources</td>
<td>Monitoring Sensors</td>
</tr>
<tr>
<td>Cause</td>
<td>Identify the term which produces the given term.</td>
</tr>
</tbody>
</table>
9. Pattern Name : Clarification and restatement.
Problem Definition : To rephrase or modulate the term in someway or put the term’s focus in a slightly different way in order to give the clarity for the term.
Solution : The clarification and restatement are incremental redirecting or less often diminishing.
An incremental restatement explains what was said before but adds a little to it, a new sense, and a new dimension.
Forces : Identify the features that add more clarity to the term that is not clear, and expand the term with this detail.

Use of PATDOC patterns for generation of document

PATDOC, the document design patterns designed can be modified during the evolution of software and it can also be extended for different standards. The following block diagram shows how the design patterns are used for generating software documents automatically. The inputs extracted from the various Software Requirement Specification techniques are represented using knowledge representation scheme called frames, extended with perspective descriptors for adaptability and flexibility for the evolution of software. The contents are also conceptually and causally linked. Hence we have to use some of the available standards and abstract plan along with design patterns "patdoc" which hierarchically organizes the document and also semantic and syntactic aspects are considered for producing automatic generation of software documents at various levels.
5.10 CONCLUSION

This work explains a new methodology and approach towards automatic software documentation. While documenting, capturing information from the design inputs and organizing in a well-formed format and structure for complex application is a tremendous effort for any documentalist. Hence patterns have been designed which possess the features such as reusability, flexibility, extendibility and maintainability. The output of patterns will be used by the planner, which produces the tree structure of plans, which in turn will be used by the realization phase of Natural Language Generation to produce documents.