Creative Problem Solving for the Virtual Storyteller

Students: Niels Bloom, Joost Vromen

Supervisors: Ivo Swartjes, Mariët Theune

Abstract

We propose a design for a flexible Creative Problem Solver for use in the Virtual Storyteller domain. The Virtual Storyteller generates and tells stories similar to known fairytales, and will use the Creative Problem Solver to solve problems for both the plot agent and the individual character agents.

In this design, creativity is implemented by means of a case-based reasoning system augmented by creativity heuristics. We propose an expression language that is syntactically close to the Virtual Storyteller to express the cases, problems, and solutions.

We also propose several heuristic functions that modify the problem space, and briefly discuss methods for inventing new heuristics. Finally, we discuss the cases themselves, and we explain what cases are most suited for the system as well as how to extract them from existing fairytales.

November 2005
University of Twente

Contents

1. Project Context

1.1. Introduction

Ancient Greek poets called upon the muses to inspire them. Expressionist painters let themselves be inspired by their emotions to create amazing displays of colour and form. Throughout the ages, human creativity has been thought of as something so rare that those who find it are lauded as geniuses.

And yet, creativity is used in nearly every task we perform. Even now, writing this introduction, we need to use creativity to think of ways to convey what we want to say. Humans use creativity when dealing with everyday problems. We use it in cooking, dancing, talking, or telling stories.

The ability to think creatively makes humans into incredibly flexible problem solvers, finding solutions to problems where there seem to be none at first sight. The notion of applying creative reasoning to computer problem solvers is therefore worth a look, at the very least.

During the past few years, the Human Media Interaction group at the University of Twente has been developing a Virtual Storyteller [STO03], an automated story generation system. Just like a human writer, the program needs to make certain creative choices when developing the story plot.

In this project we will use the Virtual Storyteller as a basis for our Creative Problem Solver. We will attempt to design a creativity mechanism for the program that it can use to create novel storylines and plot twists.

1.2. The Virtual Storyteller

The Virtual Storyteller [STO03] is a multi-agent framework for story creation. In this framework, every story character is controlled by an agent keeping track of the character's personality, emotional states, and goals. These "Character Agents" act out the events of a story. They are guided by a "Plot Agent", which has the task to ensure an interesting plot worthy of a story. The Plot Agent has the ability to make suggestions to Character Agents and to make events happen in the story world. In order to keep a consistent world model between the Character Agents and the Plot Agent, the "World Agent" keeps track of the state of the story world at any given time.

When the plot has been successfully generated, it is passed to the "Narrator Agent", which transforms it into a story in natural language and uses Text-To-Speech to tell the story to the audience. The general design is shown in figure 1.1. The design of the Virtual Storyteller can be studied in more detail in the work of Rensen [REN04], Swartjes [SWA06], and Uijlings [UIJ06].

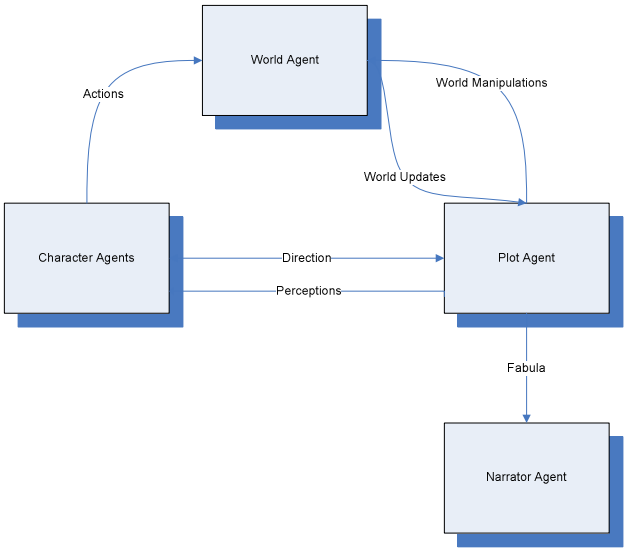

Figure 1.1: Global Design of the Virtual Storyteller

1.3. Creativity in Storytelling

In order to generate an interesting plot, both the Plot Agent and the Character Agents need to have access to some means of creativity, though each for its own purpose.

Character Agents will need to creatively think their way out of problems. Consider for example a princess locked in a room, attempting to find a way out. We would like her to think of both obvious solutions (opening the door) and not-so-obvious solutions (tying together her sheets to lower herself out the window).

Interesting stories generally contain conflict in some form or another. Therefore, the Plot Agent will often need to think of specific events in the story world that lead to conflict for one or more of the Character Agents. For example, when a hero has a goal to marry a princess, the Plot Agent could decide to form a goal to make life difficult for him. This could then lead to the Plot Agent searching for an event that would thwart the hero's goals. It could for example think of an event in which the princess is captured by a dragon, thus setting the scene for a new fairy tale.

1.4. Project Goals

We want to design a system that can be used to cater to both the creative needs of Character Agents and to those of the Plot Agent. The system should be able to find creative solutions to problems by using knowledge about the world in a new way. By designing this system, we hope to gain more understanding of human creativity, and how it can be used to make computer problem solvers more flexible.

2. Preliminary Research

2.1. Introduction

This project attempts to create a design specification usable for the implementation of a creative problem solver for the Virtual Storyteller project. In the sections below, we elaborate on our research of several publications about creativity and storytelling.

By studying the literature already available, we come to a conclusion about several key concepts. Our first task is to define what it is we mean exactly when we use the word "creative" in a computer science context. When the definition of "creative" is clear, we need to make design decisions about the general principle of our problem solver, and the way in which we will express the problems and solutions.

2.2. On Creativity

Finding a formal definition for an inherently human attribute such as creativity is a difficult challenge to say the least. The concept of creativity has previously only been thought about as a part of psychology and philosophy, and has not been seen as a Computer Science problem until several years ago. Recent research in the field of Artificial Intelligence has led to a small number of formalizations of creativity. We will look at each of these definitions in turn.

One program that uses creativity to solve problems is "Creative Julia", by Kolodner [KOL93], which is used to creatively generate meal recipes with special dietary requirements. In her research into creativity, Kolodner suggests that human creativity is often based on remembering previous experiences or stories, which quickly leads to a case-based approach to computer creativity. A case-based reasoner uses a database of previous problem-solution pairs to find solutions to similar problems that it encounters.

Kolodner states that a creative problem solver should strive towards a certain "interestingness" in the solutions it generates. Creative Julia does this by iteratively applying the following algorithm:

1) Find a solution to the original problem.

2) Evaluate the solution, using knowledge of the problem context to find its "interestingness"

3) Adapt the problem by adding, removing, or changing constraints based on the outcome of the evaluation.

4) Repeat.

For example, if the program is asked to think of a recipe using rice, it will first come up with "fried rice". It will then decide, using knowledge of the problem's context, that Chinese food is not wanted, and update its problem constraints by adding the constraint "no Chinese food"

This approach can only be considered "creative" if the set of possible solutions is sufficiently large. When this is the case, matching all possible solutions to a problem becomes too complex to process, so this method attempts to use creativity to find non-obvious matching solutions.

Of course, this will work fine in the domain of meal recipes, where most queries will yield numerous possible recipes. In storytelling, however, we do not want to use creativity to creatively select a solution from a set. Instead, we want to use creativity to generate solutions to problems that we have not seen before.

However, the situation assessment step made by Creative Julia does have some merit in the storytelling domain, since it allows a problem solver to test its ideas against "reality". What happens when this constraint is ignored is demonstrated by the next creative problem solver we will look at, called MINSTREL.

MINSTREL is one of several story-telling systems that exist today. It uses creative problem solving as its sole means of writing a story. The creator of this system is Scott Turner [TUR94], who, in his research, defines creativity as follows:

"The challenge of creativity is to find and use old knowledge in new ways to form a novel and useful solution to a problem." – Turner [TUR94, page 12]

To this end, MINSTREL, like Creative Julia, uses a case-based approach to creative problem solving, employing certain "creativity heuristics" to find new uses for old cases. The creativity heuristics define small transformations to the problem space, which are applied recursively to obtain a transformed problem description to which a solution is already known. This solution is then recalled, the transformations are applied in reverse, and the resulting solution is assumed to be a creative and valid solution to the primary problem.

This approach is used with surprising results. The challenge in writing such a system is of course defining usable heuristics; the stories MINSTREL generates often contain unexpected scenes, even with only a very small number of cases in the database. Since the problem space consists of a series of constraints on possible solutions, transforming the problem space will always lead to adding, removing, or altering one of those constraints. This approach paves the way for creative solutions, which can nevertheless be incorrect.

Turner spotted this problem as well, and even provides an example. Suppose the problem solver has knowledge of a case where a monster is hungry, and eats a princess in order to satisfy his hunger. A simple generalisation heuristic could then lead to the very creative, yet certainly unbelievable situation where a hungry knight kills and eats a princess!

Turner states that these kinds of problems are unavoidable when dealing with creative problem solving. He attempts to minimize the risk of overly creative solutions by attempting to make single transformations as small as possible. This results in a preference for solutions that are creatively "closest to" the actual problem. Minimizing the creative alteration also minimizes the risk of creative error.

Still, if problems like these are unavoidable in creative problem solving, why then do we not see them in human creative problem solving? In fact, we *do.* Who is not familiar with the simple game of fit-the-block-through-the-hole played by children around the world? A child who is still developing problem solving skills may come up with interesting but wrong solutions, such as fitting square pegs in round holes. Only after attempting the action and receiving feedback from the environment can it see that its solution was flawed, and attempt to find another.

Such a thing is not possible in the world of MINSTREL. Staying with our example of a child attempting to fit a square peg in a round hole, using a simple generalisation heuristic, it is "proven" that a square peg should fit in a round hole. After all, if a round peg fits, why would a square peg be blocked?

A real child would discover the flaw in its creativity when it received feedback from the world upon trying to push the peg through the hole. MINSTREL has no such world to test its creatively found ideas against, and so it simply defines the world as adhering to its creativity heuristics, instead of the other way around.

Realizing this, we pose the following statement about problem solvers such as MINSTREL:

To minimize creativity errors, a distinction should be made between the creative problem solver and the world model for which it solves problems. Creatively found solutions should be tested against this model, and failed solutions either discarded or repaired.

Since we cannot test ideas against the real world (the problem solver is inside a computer, after all), we will have to model the world somehow. This model should be able to evaluate creatively found solutions based upon the context of the problem. For example, the model could store all information about the story so far, and check for logical inconsistencies when the creatively found solution is added.

Like Creative Julia, testing found solutions against the world model should lead to an updated problem description. This process can be repeated until a solution is found that is both creative and fits the world model.

For example, using a separate world model that has knowledge about knights and the things they are likely or unlikely to do, Turner's example would look something like this:

1) A problem is put to the problem solver: "Find an action that satisfies the hunger of the knight"

2) The Creative Problem Solver selects a solution to the generated problem from the set of known solutions and adapts it to the current case: "Kill and eat a princess"

3) The problem solver offers this solution to the world model, which rejects it, returning the new constraint "Knights do not eat people"

4) The Creative Problem Solver adds this constraint to its problem description and searches for a new solution, transforming the previously found solution to "A knight kills and eats a rabbit"

Once a correct solution is found, it can be added to the case database for future reference. This will allow the reasoner to learn from its own creativity, in essence "experimenting" on the world model to see what works and what does not, just like a child would.

Margaret Boden [BOD95] coins the phrase "conceptual space". Conceptual spaces are made up of generative principles that describe the creative room in which the artist can form his ideas. For example, the rules in a certain game constitute a conceptual space of all possible moves. Creative thinking *transforms* this space, creating possibilities for new ideas, where those ideas were not possible before.

Boden claims that ideas are only considered truly creative (as opposed to merely "novel") when they are the result of a radical transformation of their conceptual space. This assumption then means that a truly creative computer storytelling program is nearly impossible, since the size of the storytelling domain means the conceptual spaces involved are almost infinitely complex.

This outlook obviously does not bode well for this project. However, as we have seen in other literature [TUR94] [KOL93], it is possible to express transformations of conceptual space in the form of a limited number of suitable rules of thumb. These rules, while not an exact model of human creativity, can simplify the creative process into a series of transformations that a computer can work with.

2.3. A Model for Creative Problem Solving

We have seen what creativity entails, now we need to find a way to implement this creativity in the problem solver. As we have seen, various authors have tried different methods to solve this problem [TUR94] [KOL93]

What most of these methods have in common is that they use a case-based reasoning system in one way or another. In both Turner's MINSTREL and Kolodner's Creative Julia, it is even the primary means to model the creativity aspect. Based on these

observations, our Creative Problem Solver project will use a Case Based Reasoning system to model its memory aspect, and use creativity heuristics to find novel solutions.

However, cased-based reasoning in itself does not generate creativity. For this, we need to define how to implement the heuristic functions that generate new solutions from the case base. We can discern two different methods for this.

Turner's MINSTREL uses Transform-Recall-Adapt Methods (TRAMS for short), which transform the problem space to match known cases in the case base:

"In MINSTREL, the search and adaptation processes of creativity are integrated in heuristics called Transformation-Recall-Adapt Methods, or TRAMs. Each TRAM bundles a search method with a corresponding adaptation. "Transform" takes a problem and changes it into a slightly different problem. "Recall" takes the new problem description and tries to recall similar past problems from memory. "Adapt" takes the recalled problem solutions and adapts them to the original problem." – Turner [TUR94, page 13]

Kolodner's Creative Julia [KOL93] uses a similar method:

"Creative JULIA focused on three major processes, evaluation of alternatives, and updating a problem specification." - Kolodner [KOL93, page 5]

Thus, Turner's heuristics focus mainly on matching a case in an appropriate way while searching the case base, whereas Kolodner will accept any result that meets the broader criteria of the search. Turner will then adapt his result immediately to form a solution to the problem, whereas Kolodner will need another function that judges the solution found and either accepts or rejects it. If the solution is rejected, additional conditions will have to be generated to find a better solution.

Gero [GER06] mentions the possibility of Genetic Algorithms as heuristics. In this case, as opposed to finding one solution, a large selection of possible solutions is generated. These cases are then evaluated by a heuristic algorithm that selects the best solutions from the set. From these solutions, a new generation of possibilities is generated until a best solution is selected at the end of the process. Generation is done by means of adjustments to the genetic code rather then the problem itself. The adapted code then adjusts the problem for the next generation.

Note that in all three problem solvers, modifications are made to the problem space rather than the solution space, where you might intuitively expect them. After all, we are trying to fit a solution to a problem, not the other way around. However, when receiving a problem, the solution space is still unknown, whereas the problem space is always available. Transforming the problem to match a specific solution and transforming the found solution back to the original problem circumvents this.

To implement any solution of this kind, we need three functions:

- An evaluation function

This function should evaluate the retrieved solution, together with the problem's context, for validity or interestingness.

- A transformation function

The transformation function adjusts the problem in an acceptable way. Transformations should be creative, but should not lose track of the original problem.

- An adapting function

The adapting function should adapt the case found in the database to the original problem, essentially applying the transformations in reverse.

Turner relies heavily upon his transformation functions, the TRAMs, to solve the storyteller problem, almost completely ignoring the evaluation function. He relies on the TRAMs to transform and adapt his problems in such a way that context checking can be ignored, making the process no more complex then requiring a match with an existing case after a limited number of transformations.

Kolodner on the other hand takes a completely different approach, relying heavily on her evaluation function to adjust the problem in such a way that it meets the original criteria as well as newly generated requirements. She calls this step in the case-retrieval process "situation assessment" However, many of the judgments made require either human intervention or a system with extensive knowledge about the context of the problem.

Gero too relies on an evaluation function to select the fittest samples among the solutions generated by the transformation function, but he leaves the means of evaluation open.

Turner's method bears the most resemblance to this project, and forms a solid base upon which to build our Creative Problem Solver. However, as mentioned in the previous section, it lacks a clear distinction between the problem solver and the world model. We will attempt a case-based approach to creative problem solving, using creativity heuristics to adapt the problem space in the same way Turner did. Since the Virtual Storyteller already has a fairly sophisticated world model, the required separation between the problem solver and the world model will automatically be present. By generating solutions in a way similar to Turner and then subjecting them to an evaluation algorithm similar to Kolodner's, we can have the best of both worlds. This evaluation function will have to be based on additional knowledge of the story, the motivations and emotions of the characters and what is deemed to be interesting to the tale in general.

The evaluation function will have to be a part of the world model, since the problem's context is by definition not part of the problem space, and therefore not at the disposal of the problem solver. To prevent creativity errors, the world model should evaluate returned solutions, and update the problem description if a new solution is required. Solutions that are accepted by the world model will be added to the case database, allowing the problem solver to gain more extensive knowledge about its environment with each problem it solves.

Since our Creative Problem Solver does not create the world as it goes along, but rather searches for creative solutions inside an existing story world, we cannot just return

one solution and be done with it. We want to provide flexibility to the Virtual Storyteller, allowing it to choose the solution that fits its plot or character goals best.

The Creative Problem Solver will therefore return a set of solutions for each problem posed. The Virtual Storyteller can then decide whether to use one or more of the solutions, and give appropriate evaluation feedback to the problem solver.

This difference between our Creative Problem Solver and MINSTREL poses another problem. Suppose our problem solver solves the problem of a hungry agent by telling it to eat an apple. In MINSTREL, the system could just assume that an apple was present, if necessary generating a creative solution for the question of exactly *why* an apple was at that location.

In our system, we cannot assume such a thing. Perhaps the agent has an apple, or perhaps it only has a piece of bread. The Creative Problem Solver will act as a knowledge base in the Virtual Storyteller system, returning solutions without making any assumptions as to what they will be used for. Because of this, solutions returned by the system will have to be adapted to the existing story by the Virtual Storyteller.

For example, the solution “Eat an apple” is a valid solution to the problem of being hungry, but the Virtual Storyteller could decide to generalise the apple to the piece of bread in order to achieve greater story consistency.

2.4. On Expressing Problems and Solutions

So, we will build a case-based reasoner with creativity heuristics. A logical next step is to design the protocols for expressing input, output, and cases. Since the cases and the output are both story fragments, and the problem description describes problems relevant to the same domain, it is an obvious choice to choose the same expression language for all three cases. This approach will also facilitate creating the heuristics that will transform the problem space when looking for solutions.

Turner [TUR94] represents his cases as well as his problems and story fragments in a schema based representation language called RHAPSODY which he developed himself in 1985, but never published. The basic idea in this is that story fragments such as individual story goals or events are stored in a multi-part schema in the database. When a problem is presented, it is given in the form of a similar schema with certain requirements for each position within it. Parts of the schema are uninstantiated, thus defining the problem as a problem of completing the schema.

Turner’s method seems intuitive. Certainly, the domain of stories lends itself easily for schematic representation, using actors and relations to describe the world. So, the cases and solutions can be represented using such a language. Recall, though, that we need to express the *problem* description as well. A schematic language allowing uninstantiated schemas, like Turner uses, can be used to express a range of problems. However, this range excludes problems with constraints outside of the story domain, or problems with prioritized goals. For example, one cannot pose the problem *"Provide me*

with a scene about a knight saving a princess, and, if possible, defeat a monster in the process"

In this case, designing a language capable of posing all problems in the storytelling domain falls beyond the scope of this project. Turner's schematic proposal is more than flexible enough to express the problems that the Plot Manager and the Character Agents will be facing. Thus, the method that Turner [TUR94] proposes with schemas is an appropriate base from which to start with a new representation language.

The use of the RHAPSODY language might be possible, but RHAPSODY has been specifically developed for MINSTREL. Researching its exact specification and adapting it to our own cases would take a lot of time, and in our opinion, better alternatives are available.

For example, the Virtual Storyteller internally uses OWL [OWL04], a language not unlike RHAPSODY. In the next chapter, we will discuss our design for the expression language, as well as the benefits of using OWL as a basis.

3. Story Expression Language

3.1. Introduction

The first step in creating our problem solver will be to define an expression language capable of expressing the problems, the cases, and the solutions. An ideal solution would be to find one language capable of expressing all three items. This would greatly decrease the overall complexity of the system.

Some work into expressing story structure has been done by Swartjes [SWA06]. Our expression language will largely be based on his causal model of story flow. Adhering to this model allows for easy integration of our system into the Virtual Storyteller [STO04].

Swartjes' model describes story flow as a causal network, and recognizes six main elements of story structure;

1) Internal Elements – A character's beliefs and feelings

2) Events – Events happening in the story world

3) Perceptions – Things the characters perceive

4) Goals – The goals the characters set themselves

5) Actions – Actions undertaken to reach those Goals

6) Outcomes – Whether an Action to reach a Goal was successful or unsuccessful

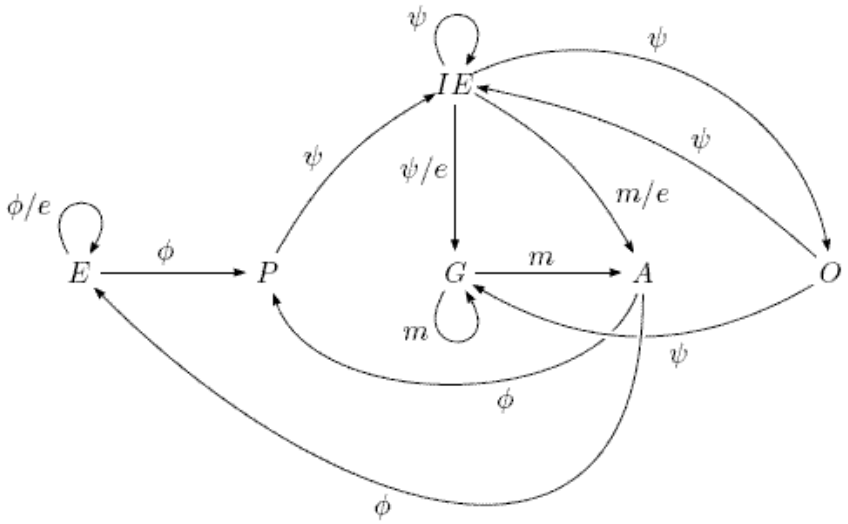

Figure 3.1: Swartjes' model of story structure. (Ψ –psychologically causes, Φ - physiologically causes, e – enables, m – motivates)

These elements should be enough to describe story plot. However, a creative problem solver will also need knowledge about the story *world*. In Swartjes' model, knowledge about the world is embedded in elements of the story plot. For example, the Goal element contains information about which character it applies to, and what state of the world the character is trying to achieve. Internally, this information is embedded in records of world states and world entities. Since we want to use creativity heuristics to transform story fragments, we will often need to work with these states and entities outside of their Goal, Event or Action context. For these heuristics to work, we need to explicitly define world states, world entities, and their relation to each other.

To achieve this, we add the elements "State" and "Entity" to the network. State elements describe world states, while Entity elements are used to describe any physical entity in the story world. These elements share no direct causal link with any of the elements in Swartjes' model, but will instead function as separate elements to which elements of that model can link.

A story structure is formed by a chain of instantiations of each of these elements. Every instantiation can have a number of properties. These properties define the specifics of the element, such as what State a Goal is trying to achieve, whether a character is alive or dead, or how hungry a dragon is.

A property value that deserves special mention is the so called "scale value". A scale value is a value that represents an interval on a scale. It defines an interval by means of a left and right border. Scale values can represent a single point on a scale by allowing the left border and right border to be equal. For example, the scale value [7..7] contains only the number seven.

3.2. Example: Thumbling

To clarify the different aspects of the language, we will use the example of the existing fairytale of Thumbling [GRIMM]. Because the entire story is rather long, we will use only a part of the story in which Thumbling prevents a theft from the pastor's house.

"When Thumbling saw that they were gone, he crept back out of the subterranean passage.'It is so dangerous to walk on the ground in the dark,' said he; 'how easily a neck or a leg is broken!' Fortunately he knocked against an empty snail-shell. 'Thank God!' said he. 'In that I can pass the night in safety,' and got into it. Not long afterwards, when he was just going to sleep, he heard two men go by, and one of them was saying, 'How shall we contrive to get hold of the rich pastor's silver and gold?' 'I could tell thee that,' cried Thumbling, interrupting them. 'What was that?' said one of the thieves in fright, 'I heard some one speaking.' They stood still listening, and Thumbling spoke again, and said, 'Take me with you, and I'll help you.'

'But where art thou?' 'Just look on the ground, and observe from whence my voice comes,' he replied. There the thieves at length found him, and lifted him up. 'Thou little imp, how wilt thou help us?' they said. 'A great deal,' said he, 'I will creep into the pastor's room through the iron bars, and will reach out to you whatever you want to have.' 'Come then,' they said, 'and we will see what thou canst do.'

When they got to the pastor's house, Thumbling crept into the room, but instantly cried out with all his might, 'Do you want to have everything that is here?' The thieves were alarmed, and said, 'But do speak softly, so as not to waken any one!' Thumbling however, behaved as if he had not understood this, and cried again, 'What do you want? Do you want to have everything that is here?' The cook, who slept in the next room, heard this and sat up in bed, and listened. The thieves, however, had in their fright run some distance away, but at last they took courage, and thought, 'The little rascal wants to mock us.' They came back and whispered to him, 'Come, be serious, and reach something out to us.'

Then Thumbling again cried as loudly as he could, 'I really will give you everything, just put your hands in.' The maid who was listening, heard this quite distinctly, and jumped out of bed and rushed to the door. The thieves took flight, and ran as if the Wild Huntsman were behind them, but as the maid could not see anything, she went to strike a light. When she came to the place with it, Thumbling, unperceived, betook himself to the granary, and the maid, after she had examined every corner and found nothing, lay down in her bed again, and believed that, after all, she had only been dreaming with open eyes and ears." - [GRIMM]

The example here contains all the aspects we will use in our language, though not all are directly visible.

Entities obviously include Thumbling, the thieves, the cook and the maid. We get a glimpse at some of the Internal Elements, which take the form of narrations the characters have with themselves or descriptions of their motivations. Note that not all Internal Elements are described in the story; some are merely implied.

Events in the story include Thumbling finding an empty snail shell as well as the encounter with the thieves. Perceptions are an important part in this story selection. In fact, it is one of the main plot mechanisms. Thumbling is trying to alarm the cook and the maid by making them have a Perception, which would alert them to the presence of the thieves.

The Goals of the thieves are rather clear, but Thumbling's Goal of thwarting the thieves is left implicit rather then being specified, as is the Goal of the maid and the cook to avoid the pastor's house being robbed.

Actions in the story include Thumbling speaking loudly to raise the alarm, the maid exploring, and the thieves trying to rob the pastor's house. The Outcomes are obvious of course. While no specific States are mentioned in the story, they are clearly present as will be clarified in section 3.9.

We will give a more detailed description of the various elements in the following sections, where we will also go into the relationships between these different elements.

3.3. Entities

The Entity element is used to define any physical objects in the Story World, including the characters themselves. There are many instances of Entities, each with its own set of properties.

For example, in the Thumbling story, Thumbling himself is an Agent, which is a subclass of Entity. The riches that the thieves want to steal are Entities as well, though with a different set of properties. An Entity hierarchy is defined in the Story World Ontology [ONT05].

This ontology defines a class tree of Entities and their specifications. Several properties of these Entities can reference other Entities. For example, the Thieves Entity can have a possessive relation with the Entity "riches".

3.4. States

We add States to the model to be able to describe a specific state of the world. These States can then be used in Goals to describe exactly what the State is the Goal is trying to attain, sustain, leave, or avoid.

An example can be found in the Thumbling story, where the thieves have a Goal to attain a State in which they have the riches from the pastor's house. This also shows that a State in the network does not always represent the world *as it is*, but often the world as characters *want it to be*.

3.5. Internal Elements

Internal Elements are elements of the Internal State of an Entity. They include the Entity's emotional state, its beliefs and other feelings such as happiness or pain. All Internal Elements are invisible to the world, as they exist only inside the Entity's mind.

An Internal Element may psychologically cause a new Internal Element. For example, a hero may believe he successfully saved the princess, which in turn causes happiness.

An Internal Element may cause a new Goal to form. Because the thieves are afraid, they form a new Goal to not be captured, causing another Goal, to get away from the pastor's house.

Goals may also be enabled by Internal Elements. For example, when Thumbling hears the thieves' plan of robbing the pastor, his Perception causes an Internal Element. This element represents Thumbling's belief that the thieves plan to rob the pastor. This in turn may enable the Goal of stopping the thieves. If Thumbling had not overheard the thieves, he could not have formed the Goal of stopping them since he did not know about it.

Internal Elements may also enable Actions. For example, if the thieves believe the pastor has a lot of money in his house, this may enable them to rob the house. If they did not believe there was anything to steal, they could not take the Action of robbing him. Whether their enabled Action is actually possible does not matter. There are no guarantees that the beliefs of an Entity are correct.

An Internal Element may directly motivate an Action. These are Actions which are not goal driven and are usually the direct consequence of an emotion. For example, when a princess is happy, this may cause her to sing. There is no Goal to sing, but rather it is a representation of her happiness. These Actions may provide insight into the Internal Elements of an Entity, but Internal Elements are never directly visible.

Finally, an Internal Element may physically cause an Outcome. For example, if Thumbling believes he has scared away the Thieves, this successfully completes his Goal of thwarting their theft.

3.6. Events

"I propose to define an Event as any change in the world that is not the direct and planned result of any character Action." – Swartjes [SWA06]

Events in the story world are often a means for story progression. As specified by Swartjes, we consider any change in the world not planned by a character an Event. Events happening in the story world can influence characters by causing a Perception. In the Thumbling example, Thumbling finding the snail shell constitutes an Event, because it was not planned. Events can also enable or physically cause other Events; consider sleeping beauty pricking her finger on the spinning wheel causing her to fall asleep.

It is important to note that Events *cannot* directly lead to Internal Elements or Goals, but must always be perceived first. A more detailed explanation for this can be found in Swartjes' work.

A hierarchy should be created in which Events are divided in a number of possible Events in the story world. Each instantiation should define its preconditions, and the effect it has on the world. For example, Thumbling knocking against a snail shell could be an instantiation of a *find_by_accident* Event, referencing the finding party and the entities found.

3.7. Perceptions

A Perception is an important step in the reasoning of characters. Events must be perceived before they can influence anyone, and Perceptions of world changes due to Actions can alter the way a character looks at the world. A Perception is the general term for anything a character feels, hears, touches, tastes, or smells.

Perceptions can be seen all through the example story of Thumbling. For example, consider the maid chasing the thieves away. The maid undertakes an Action to reach her Goal (catch the thieves) by rushing to the door. This physically causes a Perception by the thieves. The Perception by the thieves leads in turn to an Internal Element of the thieves, believing that they will be caught, prompting their Goal to evade capture.

As seen in the example, a Perception can lead to an Internal Element, which models the effect the Perception has on the character. The subject making a Perception is always an Entity; the object being perceived is always a State.

3.8. Goals

"A goal is the main drive for characters to act. Goals are desired or undesired states, activities or objects." – Swartjes [SWA06] , quoting Trabasso et al. [TRA89].

Characters try to achieve Goals. In order to do this, a Goal may motivate other Goals which must be achieved before the supergoal can be reached. Of course, Goals and subgoals do not fulfil themselves. Instead, they motivate Actions which the characters can undertake to try and reach their Goal.

Consider the Thumbling example. When Thumbling is hiding in a snail shell by the side of the road at night, he hears a couple of thieves are about to rob the pastor's house. He immediately sets the Goal to stop them from robbing the house. To achieve this Goal, the Character Agent constructs a plan consisting of two subgoals; to gain the trust of the thieves so they will take him with them to the pastor's house, and to alarm the pastor.

These subgoals each motivate an Action, the first being to tell the thieves he will help them steal the pastor's gold and the second to speak so loudly that the maid and cook awaken and force the thieves to flee.

Swartjes subdivides Goals into four different categories. A character can try to attain, sustain, avoid or leave a State. These categories apply to States (achieve, sustain, avoid, leave), Entities (obtain, keep, avoid, lose) and Actions (enable, maintain, avoid, stop). These categories are expressed as subclasses of the Goal superclass.

Any instantiation of the Goal element references the Entity that formed the Goal as well as the State, Internal Element, or Action the Goal applies to.

3.9. Actions

An Action is an action taken by a character to achieve a certain Goal. For example, when Thumbling speaks loudly he is carrying out an Action to achieve the State in which the maid is awake.

The Action can physically cause a Perception. As we know intuitively, this is because an Action can change the world which can be perceived by actors, including the one that made the Action. Thumbling's Action is perceived by the thieves, who tell him to be silent.

An Action may also cause an Event. For example, when Sleeping Beauty is spinning wool, her Action is to make thread out of the wool, but this generates an Event of her pricking herself on the spinning wheel.

Note that any accidental consequence of an Action is not an Action. When Sleeping Beauty pricks herself, she is not trying to hurt herself. This is not a failure of the Action to make thread, but rather an unplanned Event that is physically caused by the Action of making thread.

An Action does not cause an Outcome. An Outcome depends upon the Perception of the actor in question. For example, when Thumbling makes noise to wake the maid, this is a success for Thumbling's Goal to wake her, but a failure for the thieves' Goal to avoid alerting the people in the Pastor's house. Thumbling does not know that his Action was successful, until he perceives the awakened maid.

We make a subdivision of different types of Actions in a hierarchical form. Uijlings [UIJ06] elaborates on this. This division is used to specify subfields for different types of Actions and store the information about the actual Action taken. For example, a TransitMove is a subclass of Action, representing a character geographically moving to a new location.

3.10. Outcomes

An Outcome models the success or failure of a Goal. An Outcome is the result of an Internal Element; the character *believes* its Goal has been achieved or made impossible. These beliefs are the result of Perceptions, which are in turn caused by Actions.

An Action may produce different Outcomes for different Entities, such as Thumbling succeeding at waking the maid, thus causing the thieves to fail to avoid detection.

The Outcome may psychologically cause a Goal for an Agent. When a Goal fails it might be reinstated so the Entity can try in a different manner, or an entirely new Goal may be started. For example, when the thieves' Goal of remaining undetected failed, it leads to the new Goal "avoid capture".

The Outcome may also psychologically cause an Internal Element by the appraisal of the success or failure of the Action. For example, when the maid fails to find the thieves, it caused an Internal Element where she thought to have been dreaming.

3.11. Expressing Problems

Now that we have defined a language to specify world states and plot development, we need to ensure that we can use it to describe the problems that we are likely to get. The problems that the Creative Problem Solver will be asked to solve consist of incomplete schemas in the aforementioned language. If we allow elements of the schema to be *uninstantiated*, we can describe these problems.

For example, a question put to the problem solver by the Plot Monitor could be "What can I make happen to make the princess go to the forest?" This could be described in our language by an uninstantiated Event, Perception, Internal Element, and Goal, leading to the Action "Princess goes to forest".

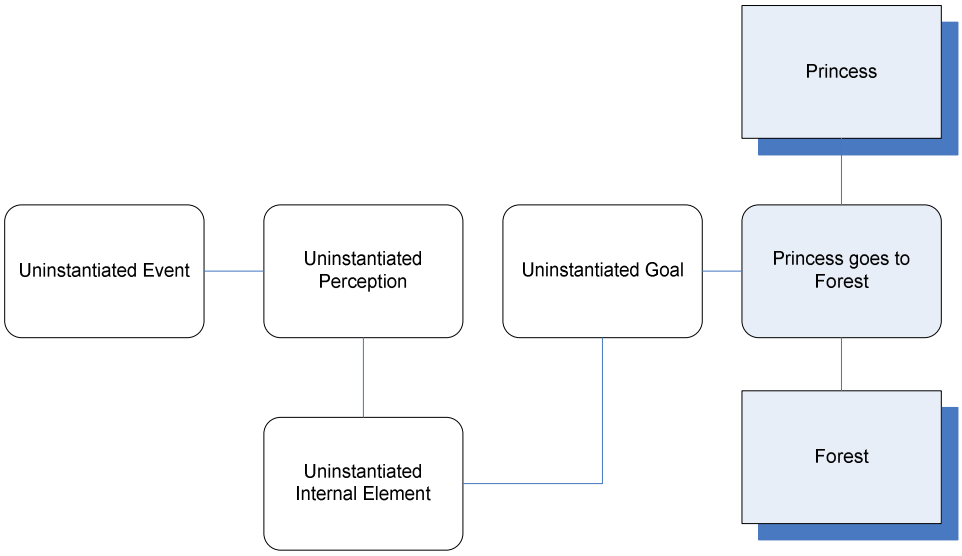

Figure 3.2: What can the Plot Agent make happen to get the Princess to go to the Forest?

The problem solver can fill in the gaps in the problem description, thus coming up with a solution.

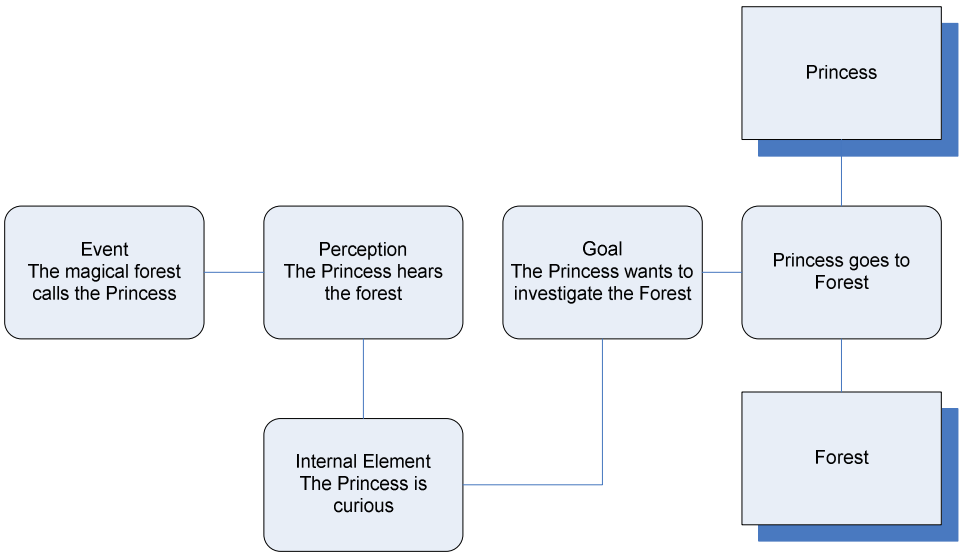

Figure 3.3: The forest is magical and calls the Princess' name!

There is a problem with this approach, though. Merely specifying the incomplete schema does not allow specific exceptions. We want to be able to pose the problem "What can I make happen to make the princess go to the forest, *without her father knowing about it?*"

To accommodate problems like this, the problem specification is divided into two parts. The first part, called *pattern space*, contains the incomplete schema that must be instantiated. The second part is called *constraint space*. Constraint space can contain any number of story fragments that must be excluded from the resulting solution. Our problem then becomes:

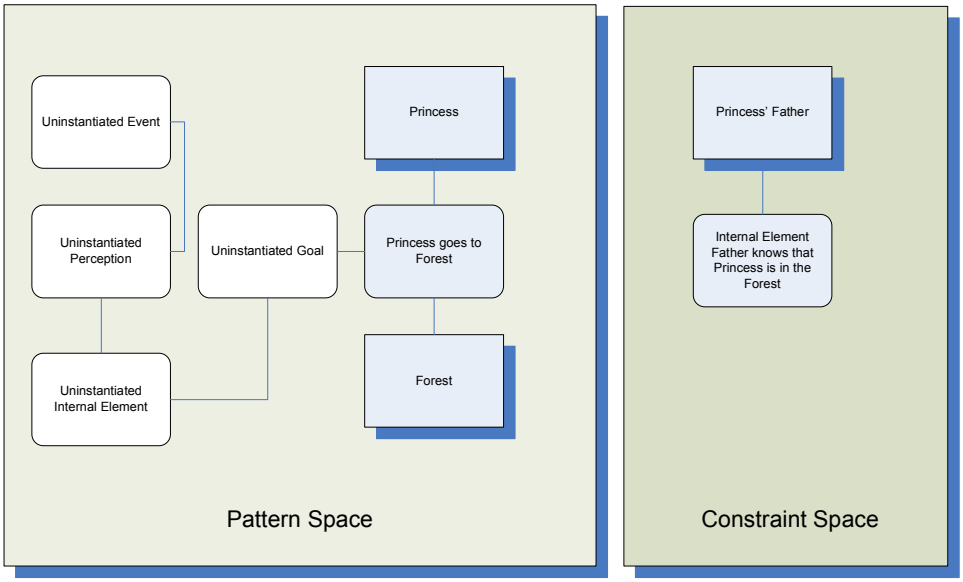

Figure 3.4: What can the Plot Monitor make happen to get the Princess to go to the Forest, without her father knowing about it?

3.12. Conclusion

We will use the language described in this chapter to graphically express the problems, cases, and solutions used in the Creative Problem Solver. To implement this language, we suggest the OWL language [OWL04]. OWL is a semantic markup language with similarities to frame-based logic. Since both the Story World Ontology [ONT05] and the Action Ontology [UIJ06] are implemented in OWL [OWL04], using OWL to implement the story expression language itself is an obvious choice.

The language is far from perfect. Of course, physical limitations do not allow for a completely exhaustive model of all possible Actions, Events, or Entities. The more complex their respective ontologies become, the more detailed the generated stories can get.

As we mentioned before, case based reasoning by itself is not extremely flexible. It needs a perfect match to a problem to successfully return a solution. In order to achieve a more flexible reasoner, we need to have a way of transforming the problem pattern. A transformed pattern will allow matches to different cases. When new matches are found, we need to apply the transformation in reverse to generate a new solution to the original problem. To allow for these transformations, we use several “rules of thumb” or heuristics.

4. Creativity Heuristics

4.1. Introduction

Creativity heuristics are used to increase the size of the solution space for the Creative Problem Solver. Creativity heuristics provide a way to transform a given problem into a different problem, and a way to reverse that transformation. This means that any solutions found for the transformed problem are also applicable to the first problem, thus increasing the size of the solution space for the original problem.

When searching for a solution, the Creative Problem Solver applies all available creativity heuristics to the problem specification. After transforming the problem, the problem solver is recursively called to find a solution for the new problem. This means that problems can undergo a series of transformations in succession. Because the resulting search tree is infinitely large, we will enforce a limit on the number of transformations. This will also prevent the transformations to the problem from stacking up too high and losing their similarity to the original.

A creativity heuristic makes a certain assumption about the validity of story fragments. A fragment is considered "valid" if it does not contain any obvious contradictions or illogical events to a human reader. When it is applied to a problem, the heuristic transforms the problem pattern to one it thinks is still valid, and tries to find a new solution.

Note that the assumption of validity is not correct in all cases. A heuristic may assume a transformed fragment to be valid, when in fact it is complete nonsense to a human reader. Consider the fragment given in section 2.2 about a creativity heuristic suggesting a knight eat a princess to satisfy his hunger! A good heuristic will make an assumption that will be valid *most of the time*. To keep errors in judgment as small as possible, heuristics should only make very small changes to the problem. This will ensure that the heuristics' combined assumptions do not stray too far from the original problem. Larger changes can be achieved by applying the same heuristic multiple times.

To illustrate the different parts of a creativity heuristic, consider the four steps that the heuristic consists of (similar to Turner's TRAM design [TUR94]);

1) Check if the heuristic is applicable to the current problem (Match)

2) Transform the problem space (Transform)

3) Match the transformed problem space to a case in the database (Recall)

4) Adapt the recalled case to the original problem by applying the transformation in reverse (Adapt)

Of these steps, the recall step is the same in every Creativity Heuristic, as it is a simple search operation on the case base. Therefore, to define a heuristic, we need to define its implementation of the match, transform, and adapt steps.

For each heuristic function, we will attempt to clarify its workings by means of one or more examples in which an actual problem is solved by means of one or more applications of the heuristic functions from a limited case base.

In his work on creativity, Turner designed heuristics for use in MINSTREL. Several of these are general enough that they can be adapted to our case with some effort. Specifically, we look at the most basic heuristics, since they are simple enough to be valid across domains.

In this chapter we will propose a number of heuristics which can form the basis of a creative problem solving program. It is by no means exhaustive but it will give a firm guideline as to what sort of heuristics will be usable in the system. In the final section of the chapter, we will discuss means of generating new heuristic functions dynamically from a case base, potentially ensuring "purer" computer creativity.

4.2. Heuristic: Generalisation

Considering we want to apply cases from different stories to the problem at hand, it is obvious we need some way to *generalise* a problem. Within the story structure, we can generalise both Actions and Entities. For example, we can generalise the fragment "The knight hits a dragon" to "The knight hits a *monster*", or to "The knight *attacks* a dragon".

Consider a valid story fragment containing an Entity or an Action. The Generalisation Heuristic states that the fragment will remain valid when one Entity or Action is replaced by a generalisation of that element.

- Match

The problem should contain at least one Action or Entity element.

- Transform

Select one Action or Entity at random, and replace it by a generalised Action or Entity from their respective hierarchies described in chapter 3, sections 7 and 10. Only generalise an element one step up in the hierarchy. Save the replaced element for the Adapt step.

- Adapt

Identify the element from the recalled solution that fits into the generalised element slot created by the Transform step. Replace all instances of this element with the element that was replaced in the original problem.

As an example, consider the following problem:

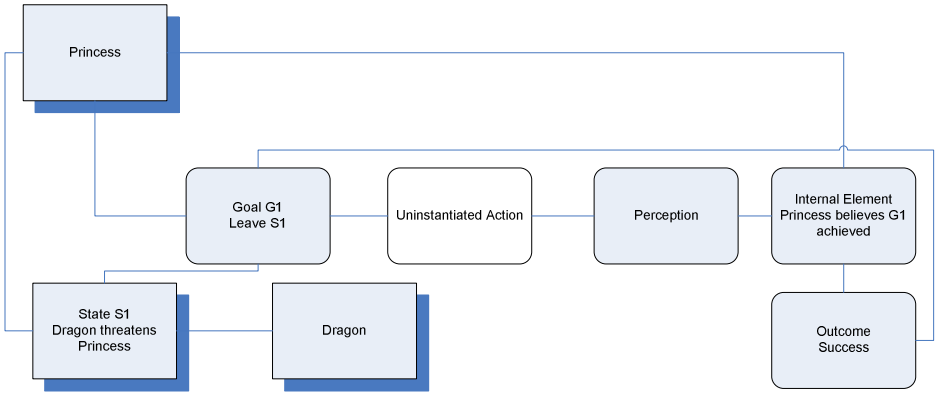

Figure 4.1: What can a princess do to escape from a dragon?

In natural language, the problem posed is "What can a princess do to escape her situation of being threatened by a dragon?"

A standard recall finds no matches in the case base, so the generalisation heuristic is applied. The heuristic checks the problem to see whether it is viable for generalisation. In this case, it is, because it contains two Entities; Princess and Dragon.

Then the heuristic picks one of the two entities at random. For purposes of our example, let us choose Princess. It then replaces all instances of Princess with a generalisation one step higher in the hierarchy, yielding the following result:

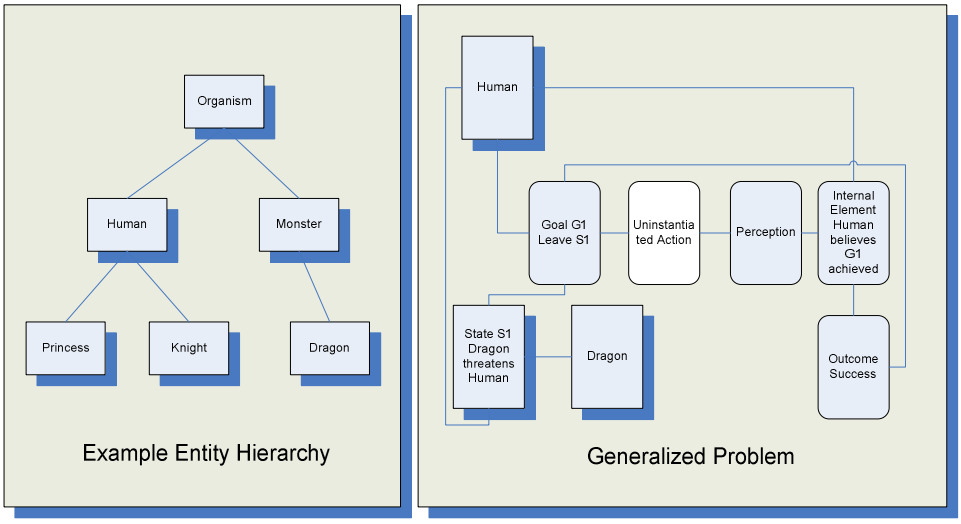

Figure 4.2: What can a human do to escape from a dragon?

Or, in natural language: “A human has the Goal to leave the State where it is threatened by a dragon.” After the problem has been transformed, a Recall returns a matching case:

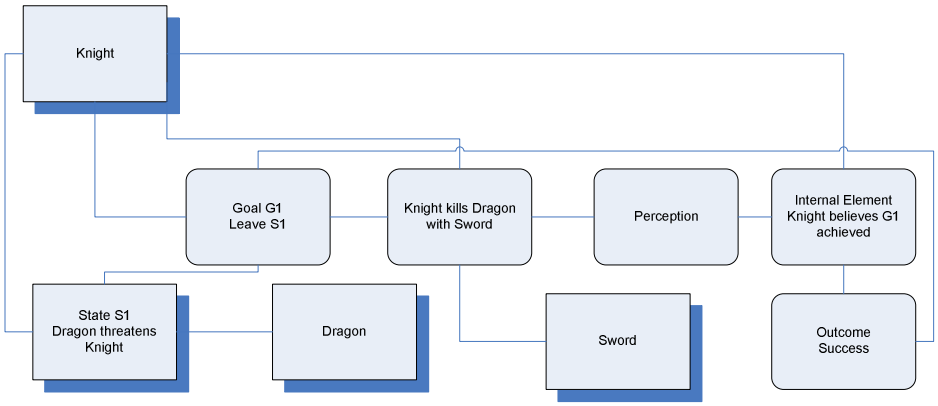

Figure 4.3: A knight kills a dragon.

“Knight kills Dragon with a sword, achieving his Goal of leaving the State where he was threatened by Dragon.” The Entity “Knight” was a successful match for the generalised Entity “A human”. In the Adapt step, the heuristic replaces all instances of Knight with the original Entity “Princess”, and returns its final solution:

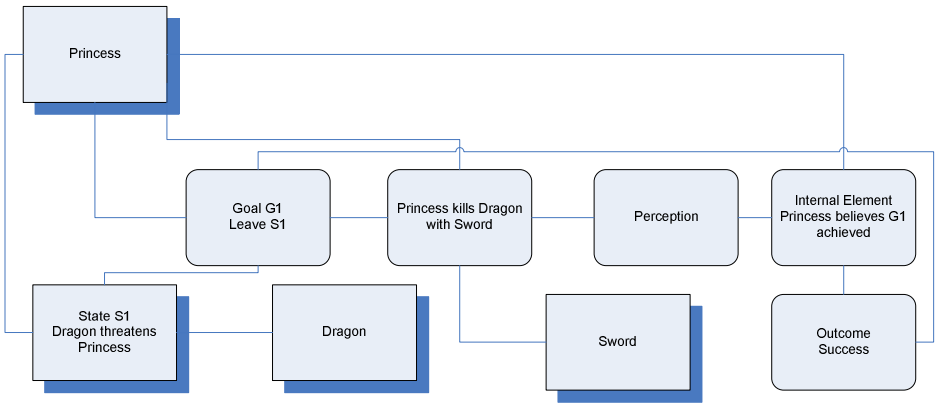

Figure 4.4: A princess kills a dragon with a sword.

Or “Princess kills Dragon with a sword, achieving her Goal of leaving the State where she was threatened by Dragon”

4.3. Heuristic: Transform Scale

Elements can have many different scale windows to store their State. For example, think of values indicating hunger, or health. The Transform Scale heuristic assumes that a story fragment will remain valid if a scale window of one of its elements is transformed in a small way.

A scale window defines a small subset of possible scale values, enclosed by a left and right boundary. The window can be transformed by shifting one or both of the boundaries to the left or the right by a certain amount.

The amount to shift should be proportionate to the number of possible scale values. The change should be large enough to make a difference, but small enough to maintain validity of the story fragment. We propose a shift of 10% of the scale size. Practical testing of the system, once implemented, should provide a better guideline.

For example, a scale ranging from 1 to 100 would allow the left or right boundaries to shift by a value of 10 per transformation. A scale consisting of five discrete entries, such as “freezing”, “cold”, “normal”, “warm”, and “hot”, would (rounded up) allow a maximum shift of 1 per transformation. To better illustrate the fact that we are dealing with scale windows, we will use integer scales in the following examples.

- Match

The problem pattern should contain at least one Element with a scale value.

- Transform

Select one of the scale values of one of the Elements at random, and replace it with a value with a slightly different interval. Save the changed value for the Adapt step.

Intervals can be changed in one of three ways:

- Change left boundary
- Change right boundary
- Shift window

So an interval of [5..8] could be transformed into [4..8], [5..9], or [4..7].

- Adapt

Identify the scale value from the recalled solution that was adapted in the Transform step. Set all instances of this scale value to the value that was replaced in the original problem.

As an example, consider the following problem, in which the hunger scale ranges from 1 to 10:

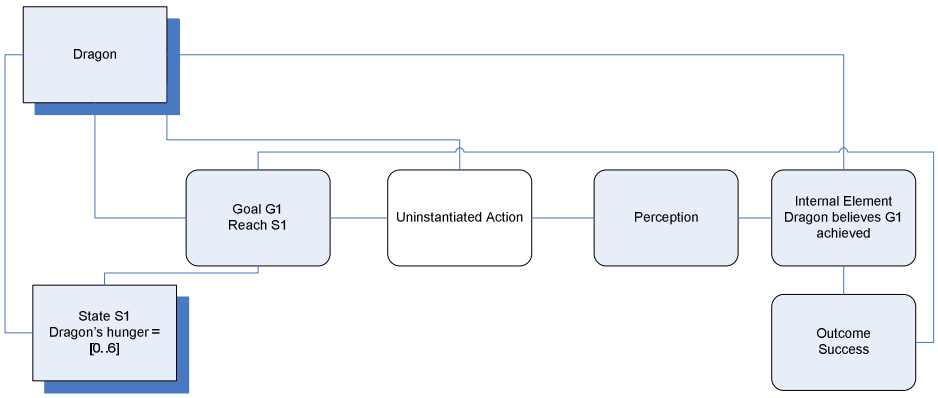

Figure 4.5: What can a dragon do to stop being hungry and feel full?

A standard recall finds no matches in the case base, so the Transform Scale heuristic is applied. The heuristic checks the problem to see whether it is viable for scale transformation. In this case, it is, because it contains a single scale value, the "hungry" property of Dragon.

The heuristic then chooses to shift the interval to the right 10%, yielding the following result:

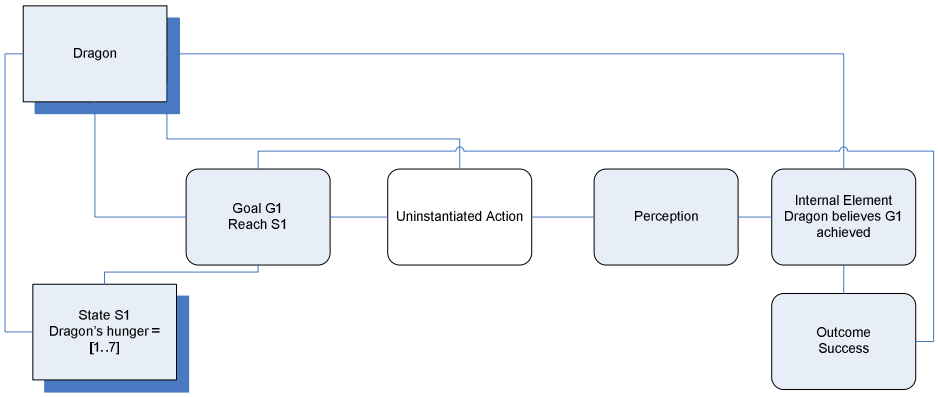

Figure 4.6: What can a dragon do to reduce its hunger?

Or, in natural language: “What can the dragon do to stop being very hungry and feel only a little hungry?” After the problem has been transformed, a Recall returns a matching case:

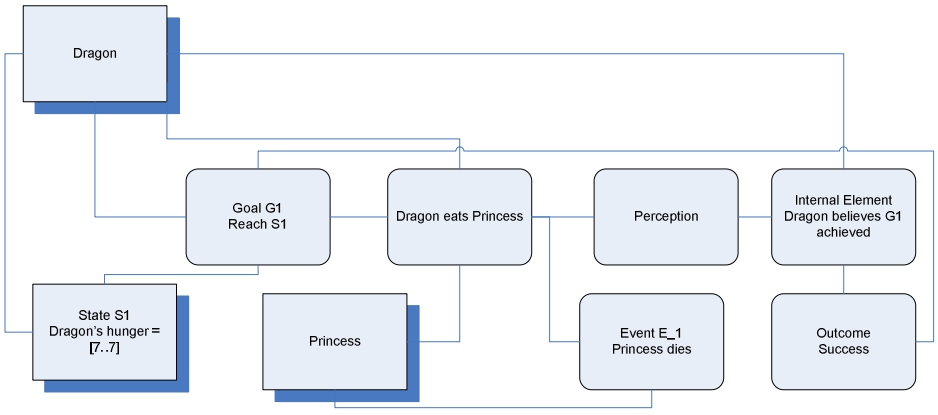

Figure 4.7: A Dragon eats a Princess, reducing its hunger to 7.

The Dragon eats a princess, achieves his Goal of attaining the State where he is just a little hungry (perhaps a princess is not a very large meal for a 50 feet dragon). In the Adapt step, the heuristic replaces the scale operation to get a scale for the final solution. In this case, the [7..7] scale is shifted 10% to the left, producing a scale of [6..6].

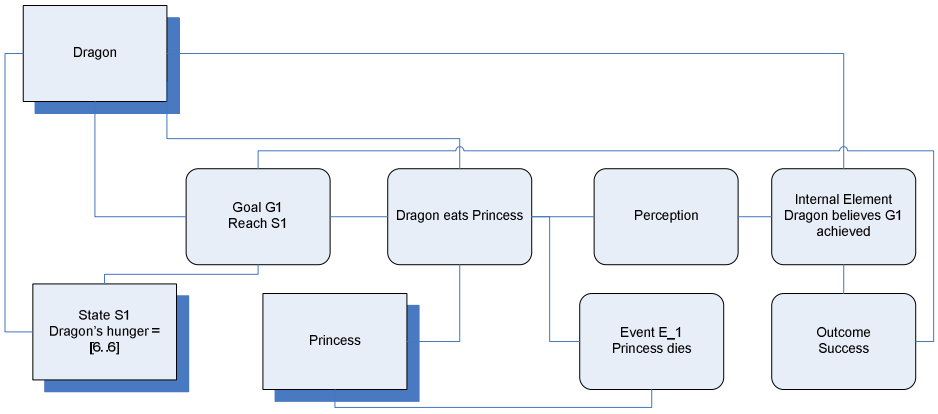

Figure 4.8: A Dragon eats a Princess and reduces its hunger to 6.

When described in natural language, this diagram represents the case “The Dragon eats a Princess and no longer feels hungry”. This is then returned as a solution to the original problem.

4.4. Heuristic: Switch Intention

Actions taken by Character Agents in order to achieve their Goals can have consequences the agent did not expect, failed to take into account or did not intend. These are represented by means of Events caused by the Action.

The Switch Intention heuristic states that the Action that caused an unintended Event is a logical action to take if your Goal is to actually reach the result of that unexpected Event.

- Match

The problem should contain at least one *Goal - Uninstantiated Action - Positive Outcome* combination.

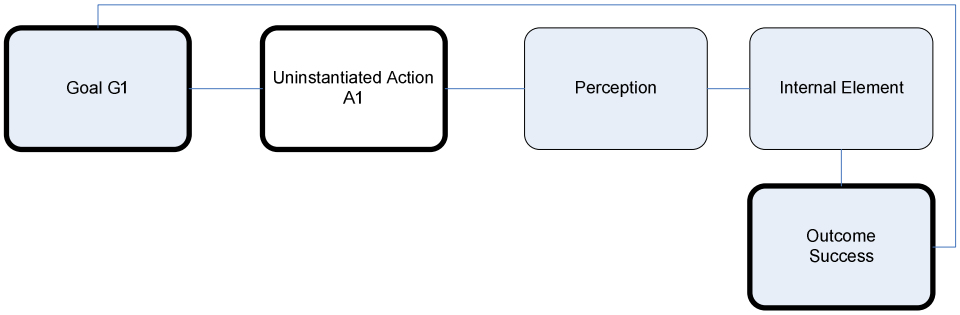

Figure 4.9: A Goal-Uninstantiated Action-Positive Outcome combination

- Transform

Select a Goal and construct an Event that achieves that Goal. For example, the Goal "reach the State where the bridge is broken" is transformed into the Event “the bridge breaks”. Construct a new problem containing a new uninstantiated Goal, prompting the original uninstantiated Action, unintentionally causing the constructed Event.

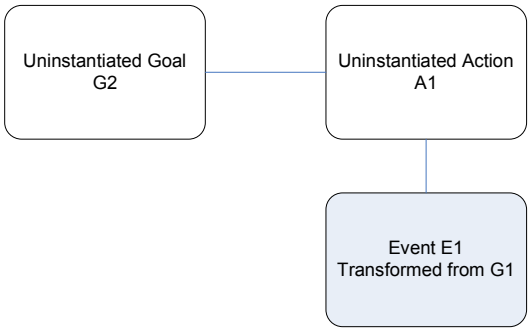

Figure 4.10: An Action taken to reach Goal G2 leads to the unexpected Event E1

- Adapt

Insert the original uninstantiated Action with the Action found in the recalled case.

As an example, consider the following problem:

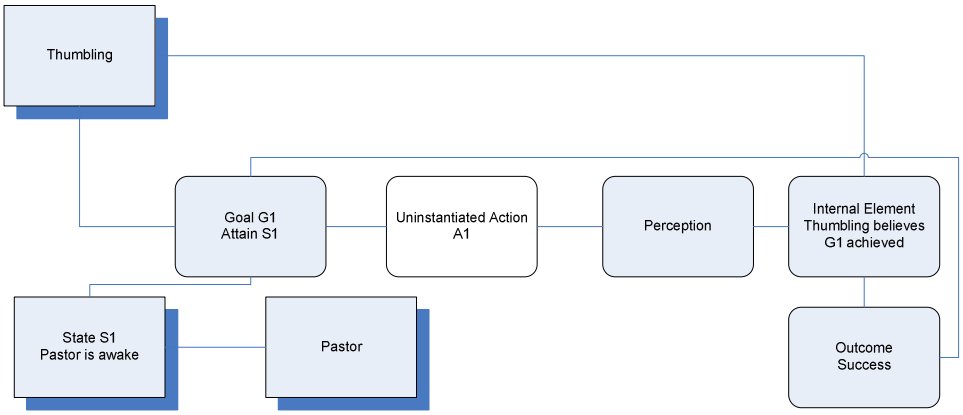

Figure 4.11: What can Thumbling do to wake up the Pastor?

In natural language, the problem posed is "What can Thumbling do to wake up the Pastor?"

A standard recall finds no matches in the case base, so the Switch Intention heuristic is applied. The heuristic checks the problem to see whether it is viable for switching intention. In this case it is, because it contains a Goal-uninstantiated Action-positive Outcome combination.

The heuristic then takes the Goal and transforms it into an Event. It constructs a new uninstantiated Goal, and links that Goal together with the uninstantiated Action and the constructed Event:

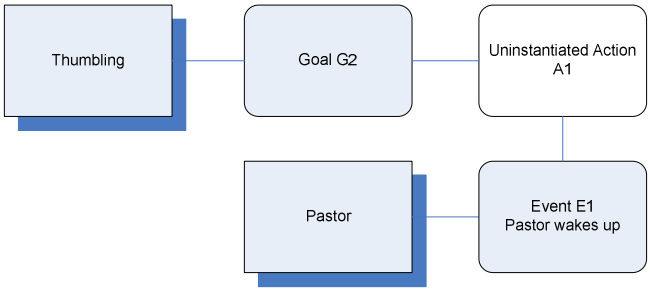

Figure 4.12: Thumbling unintentionally causes the Pastor to wake up.

Two applications of the aforementioned Generalisation Heuristic are applied to generalise "Thumbling" and "Pastor", before the system recalls the following case:

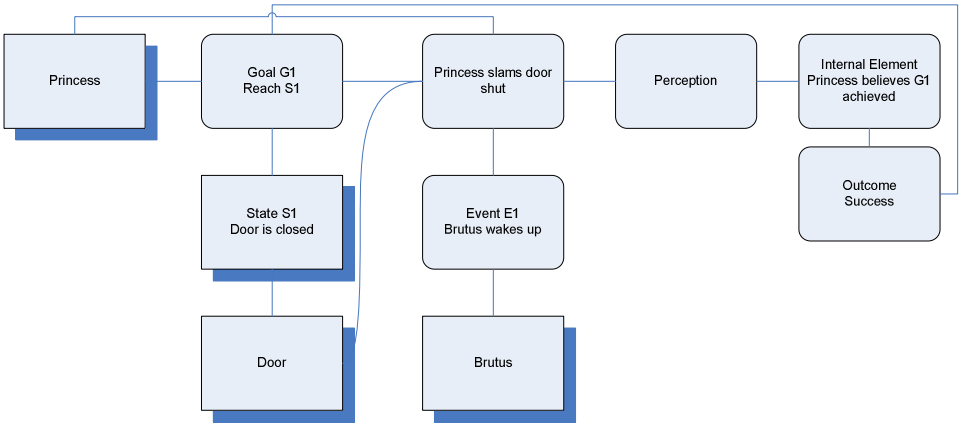

Figure 4.13: Princess slams the door, accidentally waking Brutus

Or, in natural language: "The princess wants to close the door, and so slams it shut, accidentally waking up Brutus." The two applications of the Generalisation Heuristic adapt the case to "Thumbling wants to close the door and so slams it shut, accidentally waking the Pastor". The Adapt step of the Switch Intention heuristic then takes the "slam the door" Action, and inserts it into the original problem:

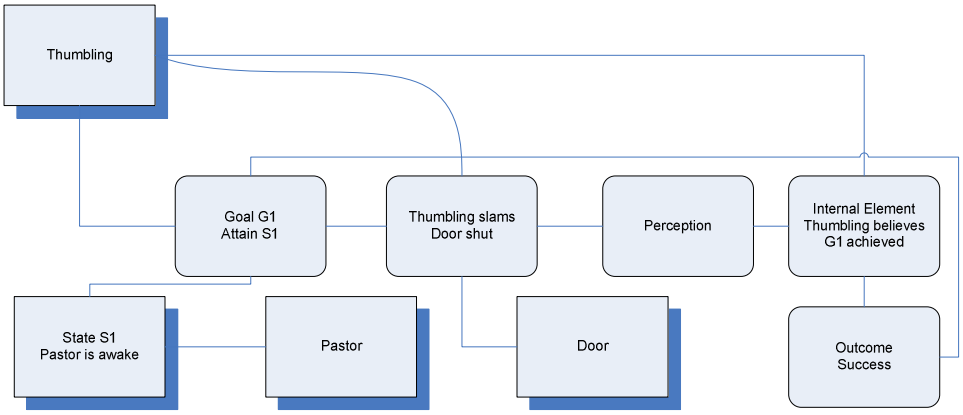

Figure 4.14: Thumbling slams the door shut to wake the Pastor

This shows the heuristic suggesting an Action that previously had an accidental consequence to be performed in order to achieve the accidental effect on purpose.

4.5. Dynamically Generating Heuristics

All of the heuristics described above are simple, intuitive rules, thought up by humans. These rules seem intuitive to us because humans build up a detailed model of how the world works during their lifetime.

For example, we can think of many cases in which the same action was performed by different people, achieving the same goal. This empirically proves the validity of the generalisation rule that those people can be generalised into one entity.

If we look at the process this way, it becomes clear that what we call creativity heuristics can also be described as patterns we can discern in our knowledge of the world. If a computer could be programmed to use pattern recognition to recognize patterns in its case base, perhaps it could postulate heuristics of its own. This would eliminate the need for creativity heuristics formulated by humans, making the computer more self-dependent in matters of creativity.

Though this promises to be an interesting topic for further research, it falls outside the scope of this project due to time and complexity constraints.

5. Constructing Cases

5.1. Introduction

The heart of case-based creative problem solving is the Case Database. The cases contained in this database form the base for every solution generated by the system.

Several details about the case database can only be determined after the system has been implemented and thoroughly tested. For example, it is hard to estimate the most efficient size of the case base, since it depends on the variety of the problems and the effectiveness of the creativity heuristics.

In this chapter, we will cover the principles behind valid cases, and present a methodology to construct new cases from an existing story.

5.2. Case Requirements

It is obvious that the quality of a solution will depend greatly on the quality of the underlying case. Of course, for this statement to be of any use, we need to know what exactly the quality of a case *is*.

The first thing that comes to mind is for a case to be *logical*. Consider the illogical case of a princess knocking over a vase to reach her Goal of being married to a prince. The case looks absurd to a human reader, but the computer will be more than happy to work with it, leading to some very strange solutions indeed.

However, a case should be more than just logical. Consider the following case, which appears logical at first sight:

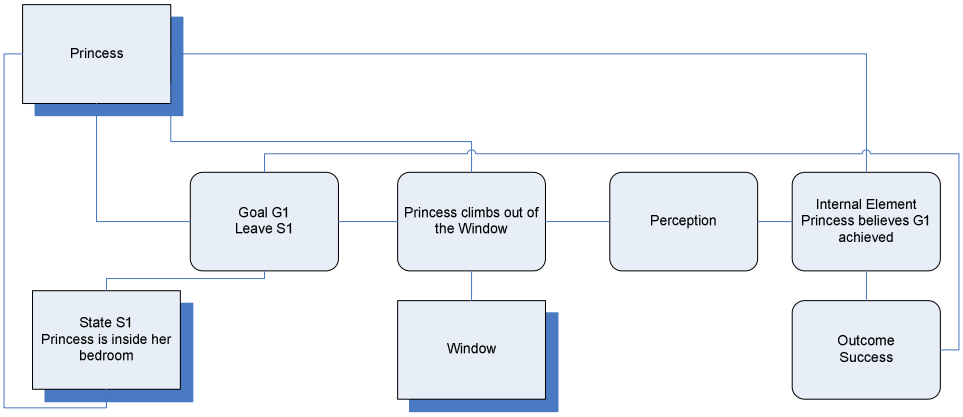

Figure 5.1: A Princess climbs out of her bedroom window to escape

This case "A princess climbs out of the window to escape from the castle" looks solid and logical. Indeed, it is a perfectly valid solution, in the original story. However, if it is used in a story where the princess is being held in a tower fifty meters high, it becomes a different thing entirely.

The problem is that the case did not specify that the window must be at ground level for the solution to be logical. This example illustrates the need for cases to be *context complete*. Cases that are context complete contain all the elements that are necessary for the solution to be viewed as "logical" by a human reader. But what about more complex cases? Once again, consider the following example:

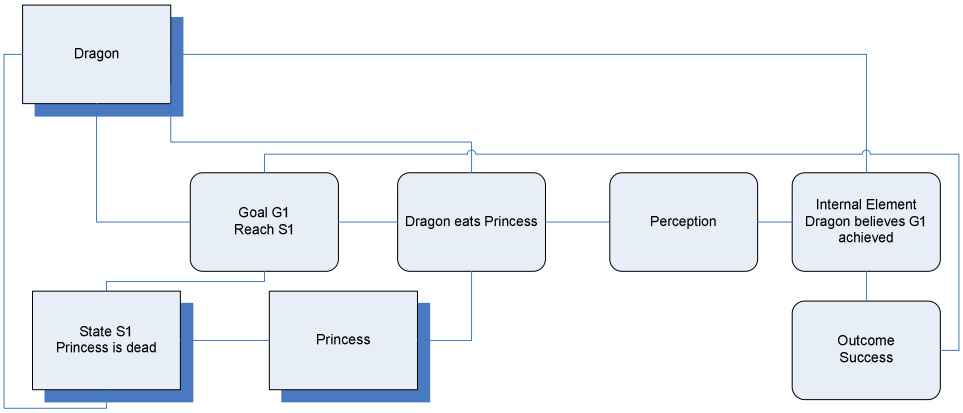

Figure 5.2: A Dragon eats a princess

This case could be viewed as context incomplete, because it does not define that the dragon should be hungry before it eats the princess, and it is illogical for a dragon to eat a princess if it is not hungry. However, if the case is used to solve the problem "how to kill a princess", the hunger of the dragon is not important, since a dragon could just decide to eat a princess to fulfil its Goal of killing her. In fact, the case is context complete in relation to "killing a princess", and context incomplete in relation to "satisfying a dragon's hunger".

To correctly include this story fragment in the case base, and allow for both problems to be solved, we would need *two* cases. One stating "A dragon eats a princess, killing her" and one stating "A hungry dragon eats a princess to satisfy its hunger, killing her". In the first case, killing of the princess will be expressed as a Goal. In the latter case, the killing of the princess will be expressed as an Event unintentionally caused by the Action of eating her.

The difference between the two cases mentioned above is the *concept* they express. One expresses the concept of "eating someone to kill them", and one expresses the concept of "eating something to satisfy your hunger." Each case should illustrate a concept, and should be context complete in relation to that concept. Note that the second

case still contains the fact that the princess dies, because the death of the princess is deemed important context to the case, even if it is not directly part of the concept.

5.3. Case Selection

We will now discuss a short story which we will divide into several different context complete story fragments. In this way we can show how a real story may be entered into the case database. Consider the following short story.

"A hungry dragon flew over the fields, searching for a princess. The princess shrieked, and hid under a bush. The dragon could not find her and flew off"

We will describe how to divide this story into three story fragments. More cases could be extracted from the story, but three will suffice for our purposes. First, we need to select the appropriate concepts to build our cases around. After these concepts have been selected, we need to determine which details to include and which to leave out.

Look at the second sentence of our example story, which says "The princess shrieked, and hid under a bush." Since it is the simplest sentence in the story, we will work with that one first. We will try to build a case to illustrate the concept "hiding from a threat". We make a first attempt to describe the sentence by means of the expression language:

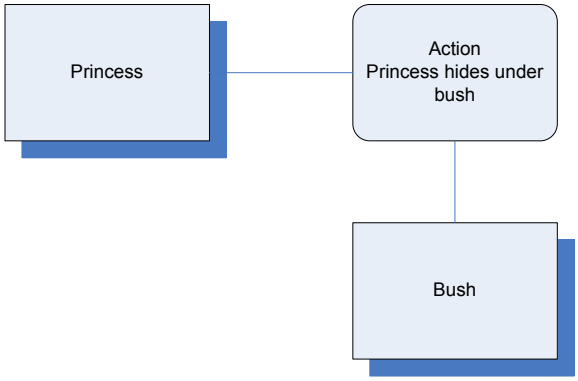

Figure 5.3: Princess hides under a Bush

While this may seem to be exactly what the sentence describes, the case misses crucial information in relation to the "hiding from a threat" concept: what is the threat? While not mentioned in the sentence, we generally assume from the rest of the story that the princess does not wish to be found by the dragon.

Of course, it could be that this is not true and the princess is in fact hiding because she is playing hide and seek with some of the local farmers or because she seeks shelter

from the rain, but since we are attempting to illustrate hiding from a known threat, we build our case as such.

So we add this motivation to the case:

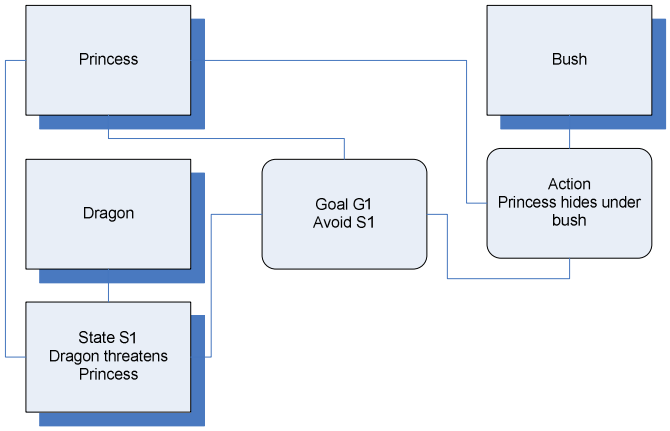

Figure 5.4: Princess hides under a Bush to avoid the Dragon

There is one more detail that we have to add. The third sentence of the story tells us that the dragon could not find the princess. Thus we can add to the case a positive Outcome. Without the Outcome, it is impossible for the system to determine whether or not hiding has any chance of being successful or not.

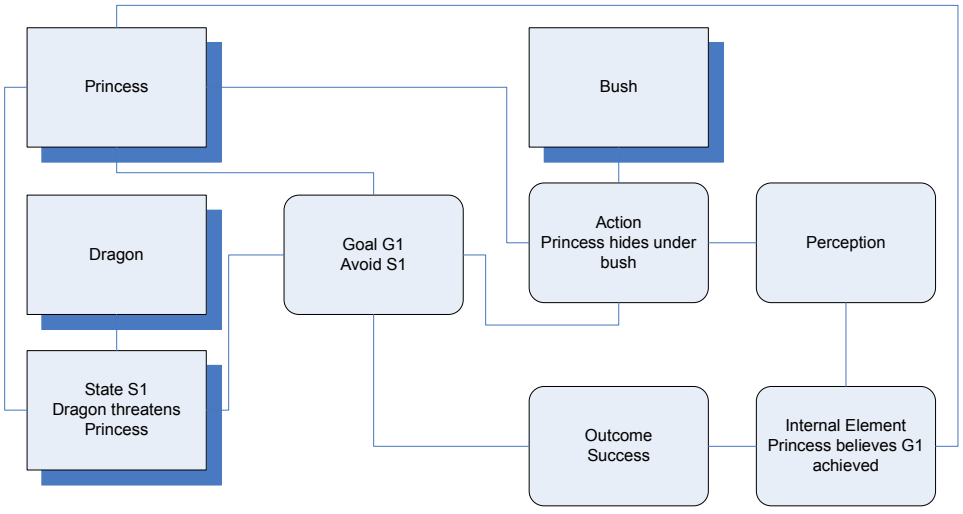

Figure 5.5: A Princess successfully hides from a Dragon

This makes the case, which now consists of two Entities, a Goal, an Action and an Outcome, context complete.

Let us now describe the simple concept of "shrieking when you are scared", found in the same sentence. A human reader assumes the princess shrieks because she is afraid. To capture this concept in a case, the reason for her fear is not relevant, so we end up with the following case:

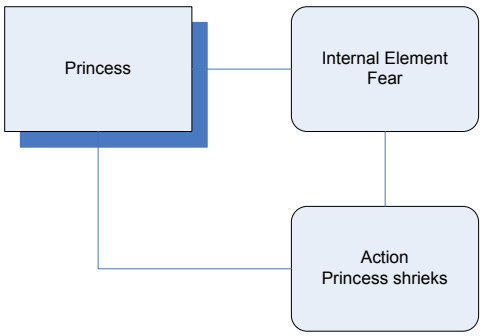

Figure 5.6: The Princess shrieks because she is afraid

The third concept we will describe is the concept of "flying over an area to search for something". We will express this concept by including the case "The hungry Dragon flies over the Fields in order to find the Princess".

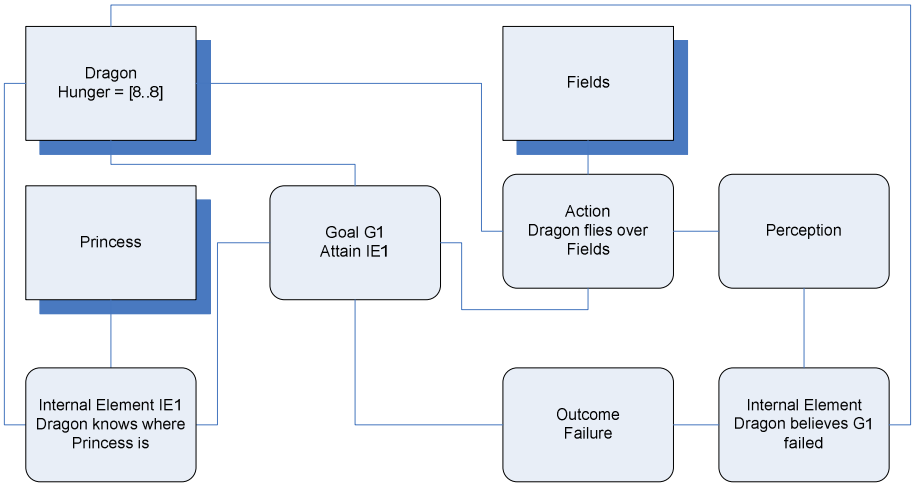

Figure 5.7: A hungry Dragon flies over the Fields to search for Princess

In this case, the hunger of the dragon is explicitly specified. In order to check if the case is not over-complete, we ask ourselves "Is the case still logical if the dragon is not

hungry?" Since the concept we chose is about searching for something, as opposed to finding a subgoal for satisfying hunger, we decide to leave out the hunger from the final case.

By contrast, we could not have taken out the dragon's Goal of finding the princess or his Action of flying over the field. These are essential parts in the case, and would have made the case illogical if they were removed.

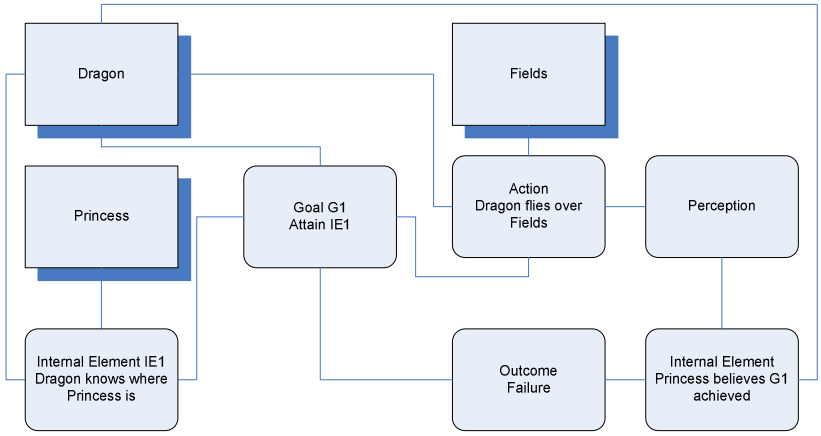

Figure 5.8: The final case

6. Future Work

This concludes our design for the Creative Problem Solver. The basic system should now be clear enough to form a solid basis for implementation. We have explained how to implement the creativity, showed in what shape to implement the design, have described examples of heuristics, and have shown how to add cases to the database.

Several things are still left undone, however. Obviously, as mentioned in section 3.6, an Event hierarchy will have to be created before any implementation of the system can be made.

Furthermore, it is important to note that there are many more possible heuristics. Concepts that come to mind are opposites (if an Action achieves a Goal, the opposite Action will thwart that Goal), co-operation (a Goal can be achieved by convincing someone else to reach it for you), and magic (a desperate measure to use if all other problem solving fails). All heuristics, including the ones we have proposed need to be tested for effectiveness in an implemented environment.

Another thing that will have to be shown by testing on a complete system is how large the case base should be. Turner [TUR94] found that a very small case base produced the best results for his design, but since ours is different, other sizes may be more appropriate.

We also suggested the option of having the computer generate heuristics from existing story fragments, something that more closely resembles true computer creativity and in our eyes merits additional research. This will require research into computer linguistics and pattern finding to produce any meaningful results.

Even with these options still open, our design is complete enough to be implemented to run the first tests on. We wish those who pick up the task of eventually implementing our designs the best of luck and hope that they will find computer-based creativity as interesting and exciting as we have.

7. References

[BOD95] *Creativity and Unpredictability*

Margaret Boden, SEHR, volume 4, issue 2: Constructions of the Mind, 1995

[GER06] *Computers and Creative Design*

John S. Gero, Key Centre of Design Computing Department of Architectural and Design, Science University of Sydney, NSW 2006 Australia

[GRIMM] *Thumbling*

Grimm's Fairy Tales as translated by Margaret Hunt in 1884. The complete text can be found at http://www.scs.cmu.edu/~spok/grimmtmp/028.txt .

[KOL93] *Understanding Creativity: A Case-Based Approach*

Janet L. Kolodner, College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280, 1993.

[ONT05] *Story World Ontology*

Jasper Uijlings, University of Twente, 2006

[OWL04] *OWL Web Ontology Language Overview*

Deborah L. McGuinness, Frank van Harmelen, W3C Recommendation 10 February 2004

[REN04] *Emotional characters for automatic plot creation*

S. Rensen et al. Proceedings TIDSE 2004: Technologies for Interactive Digital Storytelling and Entertainment, Lecture Notes in Computer Science 3105, Springer-Verlag Berlin Heidelberg, pages 95-100, 2004.

[STO03] *The Virtual Storyteller: story creation by intelligent agents*

S. Faas et al. Proceedings TIDSE 2003: Technologies for Interactive Digital Storytelling and Entertainment, Fraunhofer IRB Verlag, pages 204-215, 2003.

[SWA06] *Plot Control for Interactive Storytelling*

Ivo Swartjes, University of Twente, 2006

[TRA89] *Logical necessity and transitivity of causal relations in stories.*

Trabasso, T., Van den Broek, P., & Suh, S. Y. Discourse Processes, 12, 1-25. (1989)

[TUR94] *The Creative Process: A Computer Model of Storytelling and Creativity*

Scott R. Turner, University of California, Los Angeles, Lawrence Erlbaum Associates, Publishers, 1994. ISBN: 0-8058-1576-7

[UIJ06] *Jasper Uijilngs' Thesis*

Jasper Uijlings, University of Twente, 2006