The Cooperating Context Method

La méthode de coopération de contextes

James Hollister[1], Avelino J. Gonzalez[1]

[1] Intelligent Systems Lab (ISL), University of Central Florida, Orlando, FL, USA, JHollister@isl.ucf.edu, Avelino.Gonzalez@ucf.edu

RÉSUMÉ. Cet article présente une évaluation d'une nouvelle approche contextuelle appelée la Méthode du Contexte de Coopération (CCM) qui sert à synthétiser des récits complets qui sont intéressant et qui ont du sens pour le lecteur/écoutant. Les approches guidées par le contexte existantes sont généralement conçues pour faciliter une attention à la situation d'un agent tactique qui opère dans un environnement réel (physique ou virtuel) interactif. De tels agents sont généralement missionné pour prendre des décisions dans la poursuite de leurs objectifs et/ou en regard de l'environnement et des actions d'autres entités évoluant dans son environnement. En effet, vivent principalement dans le présent et/ou à un moindre degré, peuvent se rappeler des choses du passé ; ils se reposent seulement sur le en termes de planification pour réaliser leurs objectifs, s'il y en a. Ces approches centrées sur le contexte ne sont pas utiles pour la création d'un récit qui, par définition, n'est ni dans le passé ni dans le présent, mais plutôt dans un cadre temporel artificiel qui couvre une période de temps arbitrairement longue. CCM a été conçu et construit pour surmonter les limitations trouvées dans d'autres approches contextuelles en ce qui concerne la génération narrative automatique. CCM commence à construire la narration en examinant la situation actuelle pour créer une liste de tâches qui sont pertinentes à la situation rencontrée par l'agent. Grâce à une série d'algorithmes, la liste des contextes capables de permettre d'effectuer ces tâches est réduite à deux listes de contextes, l'une de priorité élevée et l'autre de priorité basse, tout en supprimant les autres contextes jugés non pertinents par rapport aux besoins actuels. L'ensemble des contextes les mieux adaptés à la gestion des tâches est sélectionné et les connaissances contextuelles sont utilisées pour traiter les tâches pertinentes. Tout au long de ce processus, les contextes activés et les actions prises par tous les agents (c'est-à-dire les personnages de l'histoire) sont enregistrés et font partie du récit émergent. Les applications potentielles de CCM et sa capacité à construire des récits complets comprennent la génération automatique d'histoires, la création de scénarios pour les jeux vidéo de tir à la première personne et la création de scénarios simulés pour l'entraînement tactique dans les opérations de premier répondant et dans les opérations militaires. L'application décrite dans cet article concerne la génération automatique d'histoires pour enfants. Des tests approfondis de CCM ont révélé que la méthode fonctionne comme prévue.

ABSTRACT. This article presents and evaluates a novel contextual approach called the Cooperating Context Method that can serve to synthesize complete narratives that are interesting and make sense to the human reader/listener. Existing context-driven approaches are generally designed to facilitate the situational awareness of a tactical agent that operates in an interactive world environment (whether physical or virtual). Such agents are generally tasked with making decisions in pursuit of their objectives and/or in light of the environment and the actions of other entities in the environment. In effect, these agents live mainly in the present and/or to a much lesser degree, can remember things from the past; they only relate to the future in terms of making plans to achieve their objectives, if any. These existing context-centric approaches are not useful for the creation of a narrative which, by definition, is neither in the past nor the present, but rather in an artificial time frame that covers an arbitrarily long time period. CCM was designed and built to overcome the limitations found in other contextual approaches with respect to automated narrative generation. CCM begins to build the narrative by examining the current situation to create a list of tasks that are relevant to the situation being faced by the agent. Through a series of algorithms, the list of contexts that are able to perform these tasks is narrowed down to two lists of high priority and low priority contexts, while removing those other contexts deemed irrelevant to the current needs. The set of contexts best suited to manage the tasks are selected and the contextual knowledge is utilized to address the relevant tasks. All along this process, the contexts activated and actions taken by all agents (i.e., the characters in the story) are recorded and become part of the emerging narrative. Potential applications of CCM and its ability to build complete narratives include automated story generation, building scenarios for first person shooter video games, and creating simulated scenarios for tactical training in first-responder operations and in military operations. The application described in this paper is for automated generation of children's stories. Extensive testing of CCM revealed that it performs as it was intended.

MOTS-CLÉS. Méthode du Contexte de Coopération, MCC, Raisonnement Guidé par le Contexte, Génération Narrative, Storytelling.

KEYWORDS. Cooperating Context Method, CCM, Context-driven reasoning, Narrative Generation, Storytelling.

1. Introduction

A narrative is the description (or its telling, as in the case of oral storytelling) of a sequence of actions and events that involve one or more characters that face challenges in a world environment. Such narratives have been used over the centuries for several purposes, including teaching, inspiration, recording history, and entertainment. It is important in most cases that the sequentially-ordered events that make up the narrative display a natural continuity that makes for a plot that meets the intent of its author (e.g., teach something, present a moral, or merely entertain), yet be plausible and interesting to maintain the audience's attention.

For most of our history, narratives have been composed by human authors to suit their specific objectives. However, recently with the current advanced state of computation, several researchers have embarked on the pursuit of automated narrative generation, particularly in the form of story generation. There is a rich body of literature on the subject of automated story generation whose exhaustive discussion is beyond the scope of this article. However, suffice it to say that the field still has a long way to go to achieve fully automated generation of narratives that are equivalent in plot depth, breadth and character development to those created by human authors.

Contextual approaches can be potentially very useful in automatically synthesizing narratives. It can be argued that in life, we progress through a sequence of contexts that begins when one is born and stops when one dies, always transitioning from one context to another in between as the situation calls for and in response to our actions. Narratives can be said to be similarly composed of shifting contexts faced by the character(s) (i.e., agent(s)) and influenced by its (their) own actions, those of other agents, or merely due to natural phenomena (e.g., an earthquake, a blizzard, nightfall). In such a view then, the sequence of events that compose a story can be thought of a sequence of contexts, within which the characters (agents) make decisions and perform actions that are contextually relevant.

Existing contextual approaches include such paradigms as *Context-based Reasoning* (CxBR) [1], *Context-mediated Behaviors* (CMB) [2], and *Contextual Graphs* (CxG) [3] as well as several context-aware approaches [4] (see their brief descriptions in Section 2 below), among others. These context-centric approaches, however, have been generally designed to facilitate the situational awareness and subsequent actions of an intelligent, autonomous agent that operates in an interactive world environment (whether physical or virtual). That is, they are used to recognize the context faced by an agent and prescribe the appropriate context-relevant actions to take next in order to best address the present situation. In effect, these agents live mainly in the present and may (or may not) have limited memories of its past. Their only relation to the future is in the actions taken in the present that will affect the future while in pursuit of their objectives (if any). Furthermore, the contexts they face are largely (although not entirely) the product of the environment and other intelligent agents therein – that is, it is largely forced on them.

For these reasons, these existing context-centric approaches have not been found to be useful in the automated creation of a narrative, as the latter involves the synthesis of the complete sequence of contexts and the actions of the agents therein. Furthermore, narratives are, by definition, neither in the past nor the present, but rather in an imaginary environment where the contexts in the sequence that makes the narrative are purposely synthesized (and imposed) by the author to pursue a larger objective (e.g., teach, moralize, entertain). The agents therein are neither intelligent nor autonomous within the story, and do not seek any objectives that are not part of the narrative.

Our research described in this article has led to the creation of a contextual approach that can be specifically used to synthesize narratives automatically. This approach is called the *Cooperating Context Method* (CCM). To properly place CCM within the state of the art, we begin with a discussion of the existing works in context-centric reasoning paradigms.

2. Brief Review of the Literature

In this section we briefly review the works in context-centric reasoning that are relevant to our research. A complete review of this literature, however, is outside the scope of this article.

Work in context-centric reasoning can be roughly split in two directions: 1) the use of context to provide a human with contextually-relevant information in the role of a personal assistant in the physical world; and 2) the use of context to allow an intelligent autonomous agent to navigate an environment, possibly in pursuit of an objective, either in the physical or virtual worlds.

Context-aware computing [4] seeks to recognize the context in which a human user finds herself/himself in the physical world. This is done to facilitate the provision of services that address his/her current needs in real time, all within an automated personal assistant system. This view of context has led to significant advances, and continues to form the backbone of a highly active research community. These operate mostly in the present time of the physical world. There are many applications of context-aware computing – too numerous to describe here, and also largely irrelevant to our work.

Nevertheless, context plays another equally important role in computing – that of modeling broader human behavior to enable tactical reasoning either in the physical or the simulated worlds. For this type of application, three systems of note exist and are very briefly described next.

Context-based reasoning (CxBR) [1] bases its approach on defining *contexts* that contain prescriptive knowledge (although it is not called that in CxBR) about what the agent should do when in a particular context, as well as how to do it. Contexts in CxBR can be and typically are hierarchically organized in terms of a Mission Context at the top of the hierarchy, followed by Major Contexts and then Sub-Contexts. The higher in the hierarchy, the more encompassing the context is. Major and Sub-Contexts contain control knowledge – that is, they "know" how to control the actions of the relevant agent(s) when the agent finds itself in that context. This is typically done through functions attached to a context and/or with so-called *action rules* which are also attached to a context.

In addition, each context must also "know" when the situation has changed enough so that the conditions that enabled it to control the agent are no longer present. At this point, some other context that is more relevant to the present needs of the agent must take over control of the agent. For example, an automobile driving agent is being controlled by a context that knows how to drive on a freeway. The agent then exits the freeway and finds itself in a dense urban streetscape. Some other context that knows how to drive in urban streets then becomes active and takes over control of the agent. This is critical, as inability to effect a transition between contexts could preclude the correct behavior of the agent. Imagine the driving agent racing down a crowded urban street at 120 km/hr because it still thinks it is in a freeway. At best, it would get a traffic citation and fine; at worst, it would get into a deadly accident.

One significant feature of CxBR is that it only allows one context to be controlling the agent at any one time – the so-called *active context*. This process of de-activating a no-longer-relevant context and activating a newly-relevant context is done through *transition rules* associated with the context in control. These rules actively and continuously monitor the environment (internal and external) to detect a change in the agent's situation and initiate a transition to another context. The advantage of this is that it allows a distributed form of control – the context in control of the agent always decides to which other context it would pass the "token of active control", so to speak. Therefore in summary, CxBR is all about knowing which particular context best addresses the situation currently faced by the agent, and knowing what to do when it is in control of the agent.

Context-mediated Behaviors (CMB) [2] is similar in many ways to CxBR, but different in others. Unlike CxBR, it permits multiple contexts (called *c-schemas*) to be concurrently active. It does so by merging the features of the concurrent contexts into one new context. It also features a centralized context management system (which c-schema to use, when and how to merge two c-schema), whereas CxBR distributes that function (determining which major context to activate) to the contexts themselves.

The third major such system, albeit somewhat less similar to CMB and CxBR is *Contextual Graphs* (CxG) [3]. Although it can be used to control agents continuously over a period of time in an environment, CxGs are better suited to provide one-time decision support. CxGs seek to refine the context via question-answer or acquisition of data so as to disambiguate what course of action is appropriate for that refined context. The CxG system is like a decision tree with four built-in different contextual elements to analyze the situation and select contextual knowledge to respond accordingly. The decisions are made throughout the process by moving through the graph from the left to right, responding to questions and taking actions accordingly.

The above context-centric approaches to modeling human performance have one thing in common – they refer mostly to the present time in an environment, with possibly some memory of the past, to prescribe (or in some cases, predict) and/or select future actions/decisions by agents (human or otherwise). No system to our knowledge uses context to synthesize a complete narrative that composes its own environment and selects the actions performed by agents therein.

CCM was inspired by both CxBR and CMB. CxBR, while seeking to maximize simplicity, allows only one context to be active at any one time. So, the transition to another context is simple but must be unambiguous, so it picks the most relevant. CMB, on the other hand, allows two contexts (c-schemas) to become merged into a new context when both are relevant. CCM borrows from each by allowing contexts to *cooperate* with each other concurrently to address a problem, but without merging. While contexts in CCM are not agents per se, their cooperation is not unlike that of cooperating agents in a multi-agent system. The different contexts in CCM cooperate to address tasks required to be executed to solve the current problem being faced. Some of these contexts can provide strong functionality to address the tasks (i.e., can address more than one task), while others can only address one task, and yet others do not address any. This is in some ways similar to Gonzalez and Saeki's *Competing Context Concept* (see [13]), except that in the competing context concept, the contexts compete to become the next active context to control an agent. In our Cooperating Context Method presented here, there are no agents to control – just tasks defined by the process embedded in our application to automated story generation – to extend the narrative one event at a time. However, it can be any other application-specific external process. There may be several tasks to be addressed, and the contexts cooperate to jointly address as many of these tasks as possible. This is why we named our method, the Cooperating Context Method.

Our work focuses on finding an effective method to synthesize narratives automatically. CCM is the result of our work. We describe it below in detail and report our rigorous tests to assess its effectiveness. We have applied our work on CCM to automatic generation of quest-type stories of fantasy, which involve a hero traveling long distances to seek an item, perform an action (e.g., save a princess, destroy a monster) or acquire knowledge that may be used to better his/her life and/or those of others. The intended application of our work is creating children's bedtime stories automatically given their particular preference for fairy tales. We continue by first formally describing CCM in section 3, followed by a detailed example of its operation in section 4.

3. Description of the Cooperating Context Method

CCM first plans out a complete narrative by creating the story outline and expressing it as a storyboard. After completing the entire outline/storyboard, it subsequently generates the natural language text that can be used to relate the story to the listener/reader. The storyboarded output contains the relatable events that make up the narrative. Behind these events are the contexts that enable them in the narrative. As a whole, the narrative generated must be coherent, so that if an action occurs, it is deemed possible (and plausible!) in the current or in the subsequent context. For example, if the agent is climbing a snow-capped mountain, the narrative cannot portray it digging for a buried treasure on a deserted beach in the next event (unless, of course, a flashback is involved, which at the current time is not within the capabilities of CCM). The agent must first climb down from the mountain, go to a beach and acquire the appropriate excavation tools before it can dig for the treasure. CCM ensures that this occurs.

CCM performs its functions by mimicking the human thought processes as supported by research into how individuals utilize context and contextual learning in memory [5]. This is generally accomplished by breaking down the current situation faced by an agent into a set of tasks that it needs to complete during a cycle (a single execution of the CCM process), with the final goal of using the knowledge found in the applicable contexts to perform those tasks. These contexts are selected based upon how well they can assist in addressing a stated task; that is, what functionality could a context contribute that will help the agent accomplish the task in question?

CCM was designed to utilize central processing as seen in CMB, and context transitioning (as in CxBR) to create a system that can quickly and effectively produce narratives. A major difference between CCM and the three contextual modeling approaches discussed in section 2 is the way each model views context: recognizing the current context being imposed on the agent by the "world" in order to appropriately manage an agent's actions vs. synthesizing a narrative by purposefully stringing together a sequence of contexts that contain the functions (knowledge) to accomplish the required tasks that lead to the objective of the story.

Another major distinction between CCM and the other contextual approaches discussed is the use of recursive behaviors. CMB and CxBR have no means of producing recursive behaviors. The *activity nodes* within CxG do allow a contextual graph to have some recursive behavior, but this was never the intended use of an activity node. Rather, activity nodes were added to allow a contextual graph to be scalable through abstraction. CCM explicitly allows recursive behaviors.

3.1. *Definition of Terms Used in CCM*

Here we define the terms to be used in our description and describe their functions.

Context: The term "context" has been defined in many different ways in the literature – too many to discuss here. As it pertains to CCM (and also CxBR and CMB), however, we define it to mean "*a module of actions (i.e., functions) and expectations that are relevant to a specific "situation" (see below) in the environment. Therefore, contexts contain tasks to be addressed as well as the functionality to accomplish them successfully in light of the conditions of the situation.*"

Situation: Like context, the term "situation" has been defined in many ways in the literature. We define it here as "*the set of conditions that define the specific needs of an agent that finds itself in those conditions. The situation here means the conditions that are relevant to the mission of the agent.*" Same is the case for CxBR and CMB.

Tasks: Tasks are actions to be performed by the agent to address a situation in which it finds itself. The applicable context not only contains these tasks but also how to perform/accomplish these tasks. Same is the case for CxBR and possibly CMB.

Cycle: A cycle refers to one iteration of the CCM process loop. A cycle involves a compilation of relevant tasks, search for applicable contexts and execution of functions within these contexts that address the relevant tasks.

Context List (CL): A list of all of the contexts in the CCM application – that is, the universe of all contexts that will be relevant in the overall genre of the narrative to be generated. This list is created by the developer at development time in advance of the system execution and does not change throughout the execution of CCM.

Knowledge Base: The knowledge base contains the elements relevant to the narrative to be created. All characters, locations, possible environmental events (e.g., rain, snow, nightfall), and artifacts are stored in the knowledge base. Note that the CL is not part of the knowledge base.

Task List (TL): The TL contains the tasks that CCM determines to be necessary to address the current situation in the story. It is created when CCM analyzes the current situation faced by an agent (character) in the narrative.

Active Context List (ACL): The ACL is a list of contexts that are currently in the process of being executed in the current cycle. This list is populated during the search stage of the CCM in the <u>prior</u> cycle.

Contextually Relevant List (CRL): The CRL contains a list of contexts that have been deemed appropriate (i.e., relevant) for the <u>next</u> cycle of execution. These contexts are examined as solutions to the situation in the next cycle based upon the contexts that are currently active in the current cycle. Thus, the CRL is populated at the end of current cycle using the contextual knowledge contained in the contexts presently being executed. This knowledge is about to what next contexts could be consistent with the current one.

Contextually Irrelevant List (CIL): The CIL on the other hand, contains a list of contexts that have been deemed inappropriate (irrelevant) for the <u>next</u> cycle of execution. The CIL is populated concurrently with the CRL above.

Search Context List (SCL): The SCL is created during the initial search. This list contains the contexts to be further investigated during the current CCM cycle for their relevance in the next cycle. It is populated by removing from the CL those contexts found on the CIL; thus, the SCL may or may not contain all the contexts found in the CL.

Initial Search: Refines the search context list into the high priority (HPL) and low priority lists (LPL).

High Priority List (HPL): The HPL contains a list of contexts that are known to be able to help with the current task list and <u>are</u> currently listed in the CRL. This list is created during the initial search.

Low Priority List (LPL): The LPL is a list of contexts known to be able to help with the current known task list and <u>are not</u> presently listed in the CRL. This list is created during the initial search.

Deep Search: Utilizes the initial search results to create a list of contexts that are best suited for the current situation

Future Active List (FAL): The FAL contains a list of contexts to be executed during the next execution cycle of CCM. This list is created during the deep search process. The difference between the FAL and ACL is that the ACL for the current cycle has already been executed. The contexts within the FAL have not been executed as of yet and is the result of the deep search.

In a brief review, CCM analyzes the current stage of the story to create a task list. Then CCM searches its knowledge base, which it divides into contexts that can help (contextually relevant) and those that do not help – and can impede (contextually irrelevant) - the creation of the story. Two searches (initial search and deep search) are invoked to determine which contexts are to be executed in the current cycle and their order of execution (active context list). CCM then takes the active context list and re-sorts the contexts in the knowledge base as those that can help (contextually-relevant) and those that impede (contextually-irrelevant) the story in the next cycle. CCM then performs the entire process again to extend the story to its subsequent stage. We discuss this process formally below, but first we describe the contents of a context in CCM.

3.2. *Contents of a Context in CCM*

A context in CCM contains three pieces of knowledge: 1) *descriptive knowledge* that describes the context to the system*; 2) prescriptive knowledge* – functions that perform the actions of the context*;* and 3) *contextual knowledge* that contains knowledge about relationships to other contexts. These are described in detail next.

3.2.1. *Descriptive Knowledge*

The descriptive knowledge describes to the system what knowledge is contained in a context. Each context contains a list of tasks that it is able to address. Two functions help describe the context to the system. The first receives the list of tasks (TL) and returns a Boolean **true** if any of the tasks in the TL are able to be addressed by the context. The second function also receives the TL but returns a numerical score that reflects how well the context is able perform the tasks in the TL. This numerical score is calculated independently within each context and it is based on the task. Most contexts in the CCM prototype use a scoring method that adds points for the tasks that the contexts can address and subtracts points for tasks it cannot perform. A scoring function from a context compares the TL to the able-to-solve task list in the context. The score begins at zero. For each matching task, three points are added to the score and a point is subtracted for each task that does not match.

3.2.2. *Prescriptive Knowledge*

The prescriptive knowledge of a context allows the agent to perform whatever actions the context prescribes for the agent. For example, the knowledge of how to select an adversary that will best fit the narrative analyzes each potential adversary in the knowledge base. Another example of prescriptive knowledge is the action for generating the text of the narrative at a specific point in the narrative. This knowledge is relevant to the task(s) that the context is designed to accomplish.

3.2.3. *Contextual Knowledge*

The third piece of knowledge is the contextual knowledge, which contains information about relationships to other contexts. There are two types of contextual knowledge: *contextually relevant* and *contextually irrelevant.* That is, what other contexts are relevant/irrelevant if they are to follow this one. This knowledge is stored within each context in two separate lists - one for the contextually relevant and the other for the contextually irrelevant. These lists are created by the developer during the creation of the context. During the CCM process, this knowledge is used to populate the CRL and CIL. CCM evaluates only those contexts deemed relevant for the next cycle, thereby allowing it to execute more rapidly by focusing on contexts deemed to have a likelihood of success.

3.3. *The CCM Process*

Figure 1 illustrates the overall CCM cycle and Figure 2 shows the overall (aka, main) CCM algorithm. The main CCM algorithm contains many steps - some of these steps have their own individual algorithms. The main CCM algorithm is discussed first to provide an overview. This is followed by a discussion of the four individual algorithms that compose the CCM.

The main CCM algorithm begins by defining a list of all the contexts (CL), the contextually relevant (CRL), the contextually irrelevant list (CIL), and the active context list (ACL). The CL is created once by the developer and lists all the possible contexts that pertain to this genre/type of stories. Initially, only the CL is populated. The CRL and the CIL are populated by an algorithm that updates these lists. The active context list (ACL) is populated after the deep search has identified contexts to be activated.

The CCM process consists of several cycles of CCM, each advancing the story one bit at a time by progressively expanding the outline of the story towards its ending; and later, in a separate application of CCM, by generating the natural language text that tells the story. The CCM cycle (the loop in the algorithm of Fig. 2) begins by analyzing the current situation in the story (Figure 1 Block #1). This analysis results in a list of tasks (TL) that need to be addressed at that point in the story. This is done through a function that looks for key identifying features in the situation and adds the relevant tasks to the TL. At the other end, the last step in the CCM cycle is to create two lists: the contextually-relevant list (CRL) and the contextually-irrelevant list (CIL). These lists are produced by analyzing the contextual knowledge of the currently-active contexts and considering which contexts are relevant and irrelevant for the next execution cycle. (During the first cycle of the CCM process, these two lists are empty.) However, before these lists are created, there are several more steps that occur.

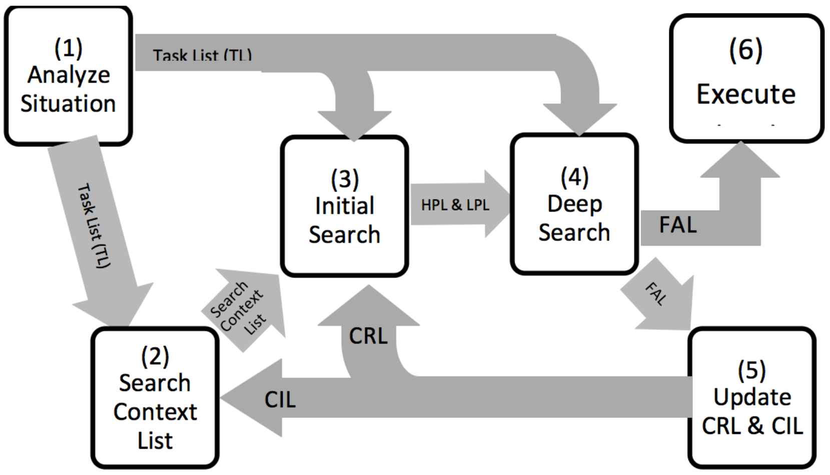

Figure 1. *The Cooperating Context Method Process*

```
1.  Define Contexts (CL) (by the developer)
2.  Define Contextually Relevant List (CRL) (during the prior CCM cycle)
3.  Define Contextually Irrelevant List (CIL) (during the prior CCM cycle)
4.  Define Active Context List ACL (during CCM execution)
5.  Main Loop
    a.  Analyze Current Situation
        i.  Define Task List TL
    b.  Create Search Context List via Search Context List Algorithm
    c.  Create High and Low Priority List via Initial Search Algorithm
    d.  Create Future Active List via Deep Search Algorithm
    e.  Move Future Active List to ACL
    f.  Update CRL and CIL via Update Contextually Relevant List and Con-
        textually Irrelevant List Algorithm
    g.  Execute ACL
    h.  Return to 5
6.  Exit Condition: Finished Analyzing Last Part of Story Outline
```

Figure 2. *The Main CCM Algorithm*

Once the TL has been created, the search context list algorithm (Figure 1 Block #2) begins with the CL and removes those deemed irrelevant (i.e., on the CIL) during the previous cycle of CCM. If the first CCM cycle, CIL will be empty and the full CL will be used. This algorithm is depicted in Figure 3.

```
1.  Given Contexts (CL) (created by developer initially)
2.  Given Contextually Irrelevant List (CIL) created during prior cycle
3.  Define Search Context List (SCL) = CL – CIL
```

Figure 3. *Search Context List Algorithm*

The next step, the initial search algorithm (Figure 1 Block #3; Figure 4), begins by calling a function from the descriptive knowledge of every context on the search context list given the task list. Figure 4 displays this algorithm. This function will respond with the Boolean value of true if the context can address at least one of the tasks. This function is how the algorithm eliminates contexts that are unable to address any of the tasks. When a context sends this value, the algorithm then analyzes the contextually-relevant list to determine whether the context name appears on it. If this context name appears on the contextually relevant list, that context is then placed on the high priority list (HPL). If it is not on the CRL, it is placed on the low priority list (LPL). Contexts that send a Boolean value of false are neglected, as they are unable to assist with the current task list.

```
1.  Given Search Context List (SCL) from Search Context List
    Algorithm (of Fig. 3).
2.  Given Task List (TL) from Analyze Current Situation
3.  Define High Priority List (HPL) created in this Algorithm
4.  Define Low Priority List (LPL) created in this Algorithm
5.  For each context C ∈ SCL
        a.  Analyze C with TL
        b.  If C returns True
                i.   If C exists in CRL
                         1.  Add C to HPL
                ii.  Else
                         1.  Add C to LPL
```

Figure 4. *Initial Search Algorithm*

Once the initial search algorithm has generated the high priority list and low priority lists, the deep search algorithm (Figure 1 Block #4; Figure 5) analyzes these lists to generate a list of contexts in the order in which their contextual knowledge should be utilized. The algorithm begins with the TL and the HPL. Each context in the HPL is now more extensively scored with regards to the task list. This is done by calling the scoring function from each context. This function is given the current task list and returns a numerical value that reflects how well a particular context can address the tasks in the TL. These functions in the contexts use a scoring method that adds three points for each task that the contexts can solve and subtracts one point for each task that it cannot solve. The context with the highest score is then transferred to the future active context list (FAL) and removed from any further deep search (because it was selected). The context's knowledge will be utilized at the end of the cycle during the utilize/execute step.

```
1.  Given Task List (TL) from Analyze Current Situation
2.  Given High Priority List (HPL) from Initial Search
    Algorithm
3.  Given Low Priority List (LPL) from Initial Search
    Algorithm
4.  Define Future Active List (FAL) created in this Algorithm
5.  Main Loop
        a.  For each context C ∈ HPL
                i.   Score C based upon TL
                ii.  Highest scoring C is moved to FAL
                iii. C's tasks are removed from TL
        b.  End Conditions: TL is empty or HPL is empty
        c.  If TL is not empty
                i.   For each context C ∈ LPL
                         1.  Score C based upon TL
                         2.  Highest scoring C is moved to FAL
                         3.  C's tasks are removed from TL
6.  End Conditions: TL is empty or LPL is empty
```

Figure 5. *Deep Search Algorithm*

The TL is also modified to reflect which tasks the selected contexts can solve; that is, tasks that are addressed by contexts now on the FAL are removed from the TL. The HPL is continually analyzed until the task list is empty or when there are no more contexts on the HPL that can address the remaining tasks.

If there are tasks still left to be completed once the deep search algorithm has analyzed the entire HPL, the low priority list is then analyzed. The LPL is analyzed until either all the tasks are completed or no more contexts remain in the LPL (i.e., no more contexts can be added to the FAL). CCM can tolerate a solution that only partially addresses the task list. If CCM is unable to locate a complete solution, it settles for a partial solution in the expectation that it may help to create a situation that can be handled in the next cycle. This allows CCM to continue execution. The FAL contains a list of contexts to be executed in order of priority of execution based on their score.

As mentioned earlier, the last step in the CCM process is to update the CRL and the CIL (Figure 1 Block #5) for the next cycle using the like-named algorithm (Figure 6). This algorithm begins by clearing the CRL and the CIL created during the previous cycle. This is done to ensure no context stays on any list longer than necessary. The algorithm then analyzes the contextual knowledge of the currently-active contexts and considers which contexts are relevant or irrelevant for the next execution cycle. This is done by examining the contextual knowledge within each context on the FAL. All the contextually relevant contexts that are listed in the active contexts list are placed in the relevant contexts list. Likewise, all the irrelevant contexts from the active list are placed in the irrelevant contexts list. The algorithm then analyzes both lists. If a context appears on both lists, it is removed from both lists. This is done to eliminate conflicting actions. The FAL then becomes the active context list (ACL) and its contexts are executed in order. After executing the ACL, the system then starts the CCM process over again in the next cycle to further develop the story outline or the text.

```
1.  Given Future Active List (FAL) from **Deep Search Algorithm**
2.  Given Contextually Relevant List (CRL)
3.  Clear CRL
4.  Given Contextually Irrelevant List (CIL)
5.  Clear CIL
6.  For each context in the FAL C ∈ FAL
        a.  If C is contextually relevant then
                i.  Add C's Contextually Relevant contexts to CRL
        b.  If C is contextually-irrelevant then
                i.  Add C's Contextually Irrelevant contexts to CIL
7.  For each context C ∈ CRL
        a.  If C exists in CIL then
                i.  Remove C from CRL and CIL
8.  Return
```

Figure 6. *Algorithm to Update CRL and CIL*

4. Example of Operation of CCM

We now provide an example that describes the CCM process in practice. Story generation is composed of two stages: 1) outline creation and 2) text generation. CCM first generates the outline through repeated cycles of the CCM algorithm. Subsequently, it uses the resulting outline in the second application of CCM to generate the text that converts the outline into a story that can be related to the listener. We focus on the outline generation process only in this example.

More specifically, in this example we utilize CCM to build the outline for a generic hero/monster encounter plot. The inputs required by CCM are: 1) The genre of the story. Here we select "medieval". 2) The length of the story. Here we select "short" (about 15 minutes long). No other inputs are necessary as CCM will make all the other selections. This short story will involve a wandering hero who encounters a village in distress and must vanquish an evil monster to save the village.

To build the story, CCM first plans out the story through the story outline. This story outline is based upon an overcoming-the-monster type of story that contains four distinct elements: the **call** stage, the **dream** stage, the **frustration** stage, and the **thrilling escape from death with the death of the monster** stage [6].

During the **call** of the story, the hero is called to save a village from the evil monster. The people in the village want the hero to help rid the village of this terrible threat. For example: In Stoker's "Dracula" the hero, Abraham Van Helsing is called to help rid the town of a vampire [7]. During the call stage, the process of creating the outline has the tasks of selecting a main character for the story (the requested hero) and selecting the monster.

The **dream** stage consists of the hero preparing for battle with the monster, trying to determine what would work best to destroy the monster. As an example Captain Ahab in "Moby Dick" appears to have things under control as he plans how to kill the white whale [8]. During the dream stage, the outline process has the tasks of selecting a secondary (bit) character, selecting the weakness of the monster, and selecting a location for the hero in where to battle the monster.

The **frustration** stage is where nothing works out as planned. There appears to be only one disastrous outcome and the hero may well fail in his plan. This would be similar to the story by Jules Verne in "Twenty Thousand Leagues under the Sea" when the Nautilus appears helplessly grasped by the tentacles of a giant squid [9]. During the frustration stage, the outline process has the tasks of selecting how the hero will fail to defeat the monster, selecting a new weakness for the monster and selecting a new location for the hero to battle the monster again.

The **thrilling escape from death** occurs during the final battle when the monster has the upper hand and just in the nick of time, the hero is able to deal a fatal blow to the monster, thereby saving the village. For example in "The War of the Worlds" by H. G. Wells, the main character is all but lost when the invading Martians begin to die by infection from earthly micro-organisms [10]. During the thrilling escape from death stage, the outline process must select an ending to the story.

4.1. *CCM in Action*

A *basic story outline* is a pre-determined story template containing the basic elements of a story type. This is not to be confused with the story outline mentioned previously, which is built upon a basic story outline and is created through CCM. In this example, the basic story outline of an overcoming-the-monster story can be seen in Figure 7.

Each element of the overcoming-the-monster story outline is filled in based upon the requirements pertaining to that specific element of the story. For example, during the hero's call, a village, a monster, a hero, and any bit characters (e.g., villagers) must be defined, as they are all present in this element of the story. CCM cycles through its body of context knowledge to fill in the rest of the story outline by deciding on the type of monster. The intelligent selection of the monster is controlled through a context that analyzes what has already been requested by the listener to make the appropriate selection.

CCM begins to create the outline by analyzing the first element of overcoming-the-monster (**The Call**) to determine what tasks need to be completed. Knowing that it is working on the outline stage, CCM generates tasks to accomplish this goal. As seen in Figure 8, CCM generated the *Select Main character* and *Select Monster* tasks.

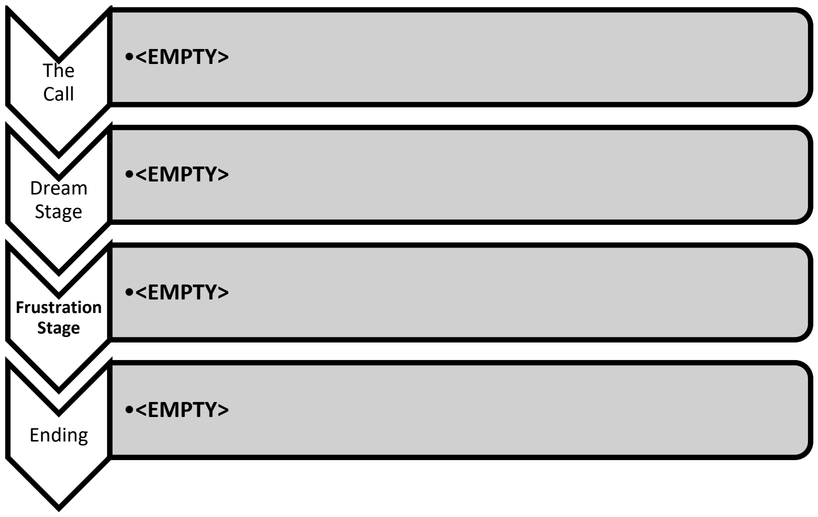

Figure 7. *Empty Overcoming-A-Monster Basic Story Outline*

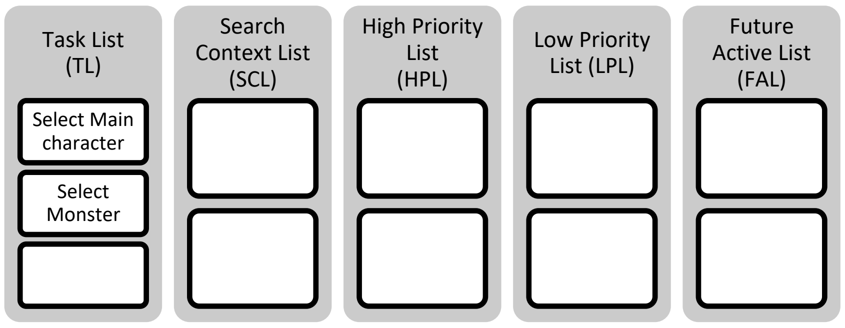

Figure 8. *Tasks List*

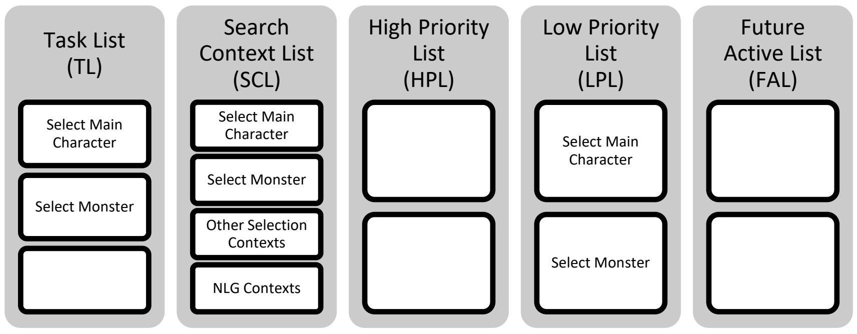

Figure 9. *Updated Search Context List and High and Low Priority Lists*

The search context list algorithm (Figure 3) generates a list of contexts to search. As this is the first time that the CCM has been invoked, the search context list is the complete list of contexts (CL), as none have been yet labeled as irrelevant by CCM (that occurs later in the cycle). The initial search algorithm (Figure 4) uses the search context list to create the HPL and LPL. As seen in Figure 9, the contexts named *select main character* and *select monster* have been placed in the LPL. They were not placed in the HPL because there are no contexts yet on the contextually relevant list (CRL). (See step 5b in Fig. 4.)

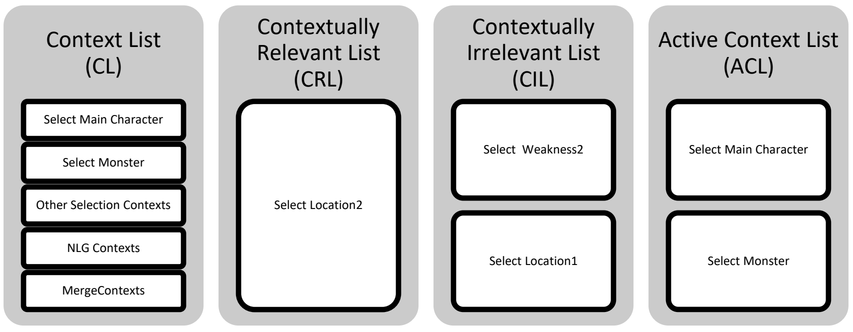

Figure 10. *Updated After the Call*

Once the HPL and LPL are generated, the deep search algorithm (Figure 5) utilizes these lists to create the future active list (FAL). Given that in this first cycle the HPL is empty, the deep search algorithm proceeds to search through the LPL, which has two contexts in it – *select main character* and *select monster*. These are transferred to the FAL and the tasks of the same names are removed from the task list (TL) (not shown in the figures). The FAL is now used by the algorithm in Fig. 6 to update the contextually relevant list and the contextually irrelevant lists (CRL and CIL). In this example the *Select Weakness2* and *Select Location1* contexts were added to the CIL because they are irrelevant to **The Call**. The updated list can be seen in Figure 10.

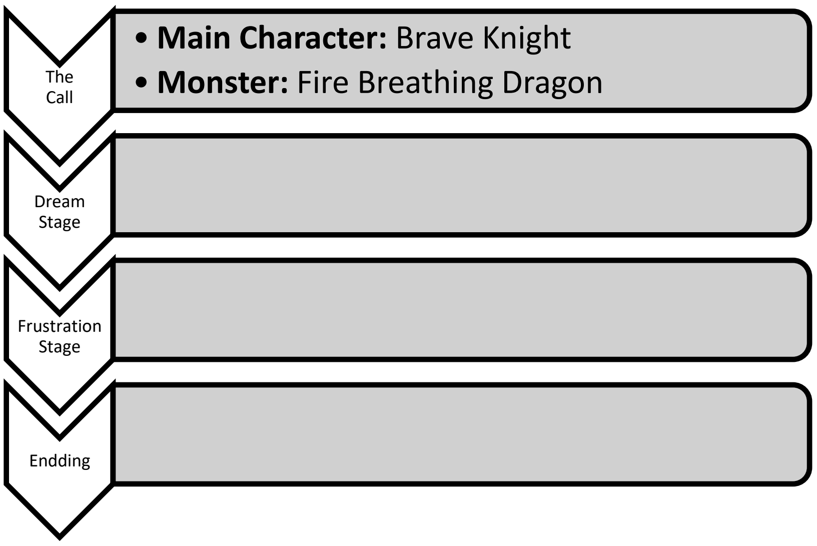
The
Call
• Main Character: Brave Knight
• Monster: Fire Breathing Dragon
Dream
Stage
Frustration
Stage
Endding

Figure 11. *Overcoming-A-Monster Story Outline with Completed Call*

After this, the contexts from the future active context list (FAL) are moved to the active context list (ACL) and they are executed. This completes one full cycle of the CCM process. During the execution, these contexts make selections from the knowledge base. A brave knight has been chosen to face a fire breathing dragon. The updated story outline can be seen in Figure 11.

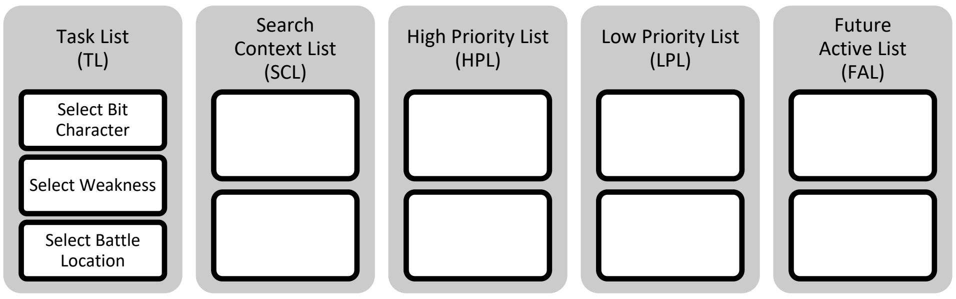

Figure 12. *Tasks List for Dream Stage*

Continuing with this example of the CCM process, the subsequent (in this case, the second) cycle of CCM analyzes the next element in the basic story outline (**Dream** stage) to determine what tasks need to be completed. As seen in Figure 12, CCM generated the *Select Bit Character*, *Select Weakness*, and *Select Battle Location* tasks.

The search context list algorithm (Figure 3) generates a list of contexts to search. As this is the second run, the search context list will be the complete list without the contexts that were deemed irrelevant from the last cycle. The initial search algorithm (Figure 4) creates the HPL and LPL. As seen in Figure 13, the contexts *select bit character* and *select Weakness1* have been placed in the LPL. The *select location* context was placed in the HPL because it was placed on the CRL during the previous cycle (see Fig. 10).

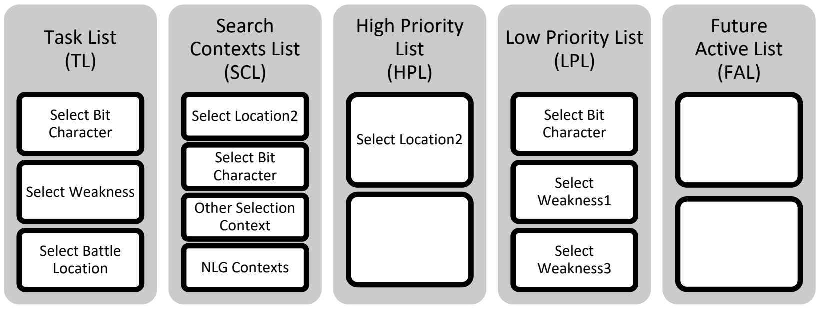

Figure 13. *Updated High and Low Priority Lists for Dream Stage*

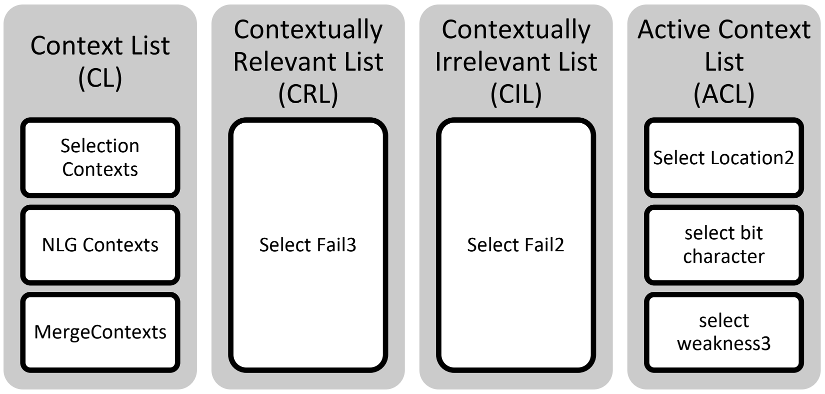

Figure 14. *Long-Term Memory Updated After the Dream Stage*

The deep search algorithm (Figure 5) utilizes these lists to create the FAL. In this example, the contexts *Select Location2*, *select bit character*, and *select weakness3* were selected. The algorithm to update the contextually relevant and irrelevant lists updates these lists (Figure 6).

The updated lists can be seen in Figure 14. The contexts from the future active context list (FAL) are moved to the active context list (ACL) and executed; this completes the second full cycle of the CCM process. Note that we have skipped the Future Active List step in our figures to save space and went directly to the ACL. The updated story outline can be seen in Figure 15. During the execution, the contexts selected states that the blacksmith would assist the hero in taking advantage of a soft belly on the dragon with the battle taking place in a forest. In battling the dragon, the hero identified the wrong weakness causing the attack to fail. The only benefit of the failed attack was that the hero did learn that the dragon has weak eyesight. This weakness is used during the final battle in a cave.

A similar process would be followed to complete the outline of the story. This is not described here but the interested reader is referred to [12] for a full description of this example. CCM continues to analyze every element until the story outline is completed. The completed outline is seen in Figure 16.

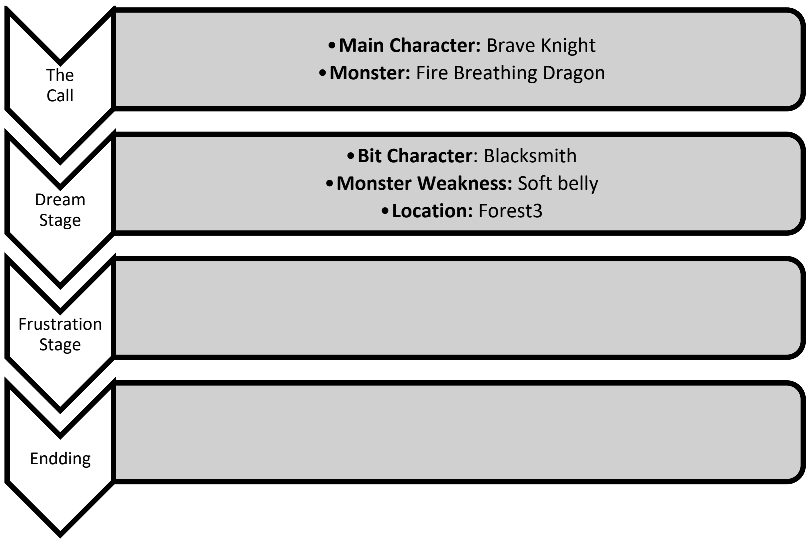
The Call
• Main Character: Brave Knight
• Monster: Fire Breathing Dragon
Dream Stage
• Bit Character: Blacksmith
• Monster Weakness: Soft belly
• Location: Forest3
Frustration Stage
Endding

Figure 15. *Overcoming-A-Monster Story Outline with Completed Call and Dream*

During the story generation phase, CCM can have multiple contexts to cover one area of the story. For example, one context can cover the hero wandering into the village, while another context will have him summoned. CCM will only select one of these two contexts to be active through the search process. This thereby eliminates the repetitive behaviors while still keeping the size of the model small.

Upon the completion of the story outline, CCM goes back to the beginning of the outline and executes the text generation stage based on the completed outline. It is in this stage that the details of the story based on the outline are generated. The result is a story able to be told by reading its text.

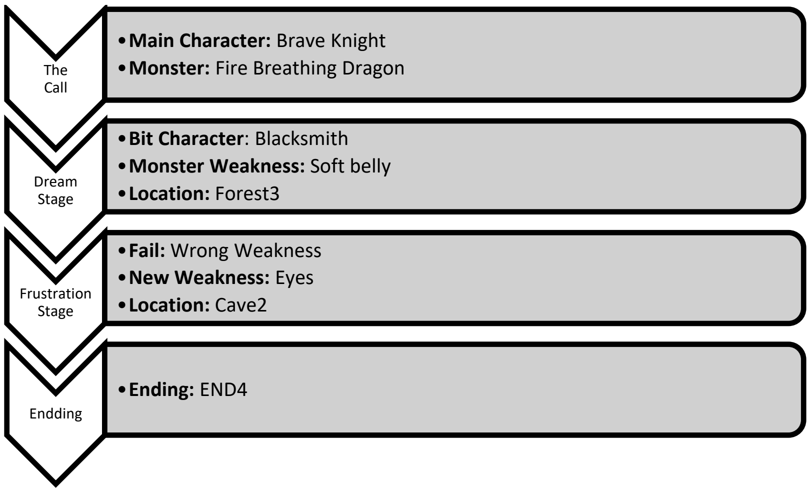

Figure 16. *Completed Outline for Overcoming-A-Monster Story*

4.2. *Strengths of CCM*

CCM is able to generate interesting and entertaining stories by utilizing contexts of the situations in a story and breaking the situation down into reasonable tasks that can be completed sequentially to generate a story that corresponds to the users' requests. This use of contexts that are relevant, yet diverse allows CCM is able to overcome the weakness of generating repetitively-similar stories by detecting differences in the contexts used to address the tasks and thus ensuring different results in story outcomes. For example, if the hero is deemed to be cowardly, he needs to behave in a timid and fearful manner.

CCM makes no assumptions of the outside world view, and by breaking the current situation down into tasks, new tasks could be generated to attempt to classify unknown items. Within the story generation, these tasks are derived from the element of the story that is currently being examined. As mentioned in the overcoming-the-monster example, the call stage requires items to be defined, such as the hero and the monster. CCM considers the stage of the story, and based upon which overall part it is working on (the outline or the text generation), it defines a task list to be completed.

A major strength of CCM is analyzing the contexts available while avoiding those that have been deemed irrelevant. This is accomplished by examining the contextual knowledge of the currently-active contexts and considering which contexts are relevant and irrelevant for the next execution cycle. This allows the search process to be flexible without using inordinate amounts of time to analyze contexts that are defined as being unimportant to the story being generated.

Another strength of CCM is its ability to give higher priority to contexts that have been deemed as good transitions. Focusing the search on tasks that need to be solved adds flexibility to the CCM process because it allows CCM to complete vastly different processes with the same contexts and to focus on what needs to be accomplished in the current context. For example, CCM can examine the same element of the story to complete the story outline the first time and then generates a text for that story element then second time it executes (clarification: Not the next cycle of execution in the same part, but rather, when CCM is applied on the subsequent text generation part). Therefore, when applied to the story generation, CCM can plan out the stories and generate the text of the stories using the same knowledge base.

5. Evaluation of Prototype

Measuring the effectiveness of CCM can only be done in the context of an application – in our case, automated story generation for children's bedtime stories. A prototype system for story generation that employs CCM was built and assessed. This section covers the evaluation of CCM through the story generation application. Specifically, we focus on the ability of CCM to create stories that are novel - that is, stories that display variability so that no two stories are similar.

5.1. *Novelty Testing*

Novelty testing follows the general procedure outlined in Peinado et al., 2010, [11] This procedure for novelty testing was intended to numerically determine a story's novelty by computing the distance between two different stories. The concept behind novelty testing is to determine how different Story A is from Story B by analyzing several different elements of each story. Story A is used as a benchmark and any changes and additions to the story are found in Story B. With the procedures and formulae determined by Peinado et al., one can calculate a linear distance between two stories that is indicative of how different these stories are. A large linear distance means that the two stories are very different from each other. Conversely, a small distance means that the stories are similar. Peinado et al. proposed to examine four general elements in a story to determine if and how they changed, which would affect their novelty: the *events*, the *characters*, the *props* and *scenarios*. Input values are binary – they are either present or not present.

Peinado et al. examined the physical events that would take place in a story and compared these events with events in the second story. They characterized this element into two major components: main events and secondary events. Main events cover the large actions within the story – the big picture - while secondary events cover the minor actions within the story. Within each component, Peinado et al., sought to categorize them as *new*, *changed in order*, *replaced with similar items* or *replaced with different items*. An event is new to the second story when it did not take place in the first story. The changed-in-order determines whether the same events in each story were presented in the same order or not. The replaced-with-similar and replaced-with-different events examine whether events were replaced during the story with other events – either with events that were similar or with events that were not similar. Peinado et al. weighted each category in the order they deemed was most relevant to novelty. For instance, a main event that has been replaced with a different event has the highest weighted value score, while a secondary new event will have the lowest weighted score. Peinado et al. utilized stories in their study that were well-known, such as a Cinderella–like tale as well as a Hansel and Gretel-like tale. The weighting of the system helped to determine how novel the new story was from the original well-known tale. Peinado et al. [11] used the following equations to calculate the event element.

$$Event = we_1 \times (we_{11} \times me_{new} + we_{12} \times me_{cho} + we_{13} \times me_{sim} + we_{14} \times me_{ch}) + we_2 \times (we_{21} \times se_{new} + we_{22} \times se_{cho} + we_{23} \times se_{sim} + we_{24} \times se_{ch})$$

In the above equation, all the variables that begin with *we* are weight values. No range was given for the weight value – only the guidance about the values relative to each other. In the same vein, variables that begin with *me* correspond to a main event while *se* corresponds to a secondary event. The *new* sub-abbreviations are used to represent the new events and the *cho* sub-abbreviation is the change of order. The *sim* means replaced with similar while *ch* reflects replaced with radically different. The following inequalities must hold true for the assigned weights above [11].

$$we_1 > we_2$$
$$we_{11} < we_{12} < we_{13} < we_{14}$$
$$we_{21} < we_{22} < we_{23} < we_{24}$$

The second element was the *character* element in which Peinado et al. looked at the characters that appeared throughout the two stories, and then compared these characters to each other. They separated the characters into two groups – *main characters* and *secondary characters*. The main characters are the subjects of the story. The secondary characters also play a part in the story – just not a major part. Within each group, Peinado et al., sought to also categorize them as new, changed-the-order, replaced-with-similar characters, or replaced-with-different characters. A new character was new to the second story when it did not appear during the first story. The change-of-order became true if the characters appeared in each story but in different order. The replaced-with-similar or replaced-with-different characters both reflect when characters were replaced during the story – either with characters that were similar or with those that not similar to each other. The equation below is similar to the one above except that it treats character elements instead of events. [11].

$$Character = wc_1 \times (wc_{11} \times mc_{new} + wc_{12} \times mc_{cho} + wc_{13} \times mc_{sim} + wc_{14} \times mc_{ch}) + wc_2 \times (wc_{21} \times sc_{new} + wc_{22} \times sc_{cho} + wc_{23} \times sc_{sim} + wc_{24} \times sc_{ch}$$

In the equation above, all the variables that begin with *wc* are weighted values. Peinado et al. weighted each category in the order they deemed was most important to novelty. A secondary new character has the lowest weighted score while a main character replaced with different character has the highest weighted score. Again, only guidance was provided as to the relative values of the weights in relation to the other weights. The variables that begin with *mc* correspond to a main character while *sc* corresponds to a secondary character. The *new* sub-abbreviation is used to represent the new character and the *cho* sub-abbreviation is the change of order. The *sim* means replaced with similar while ch means replaced with radically different. The following inequalities must hold true for the assigned weights above [11].

$$wc_1 > wc_2$$
$$wc_{11} < wc_{12} < wc_{13} < wc_{14}$$
$$wc_{21} < wc_{22} < wc_{23} < wc_{24}$$

There are a third and fourth elements in Peinado et al.'s method to calculate novelty – one for *props* (an item used to assist the main characters in their quest) and one for *scenarios* (important to the story because they offer new locations for the events). The corresponding equations are very similar (although with some small differences) to the ones above, so we omit them here in the interest of space and avoidance of monotony, and refer the reader to Peinado et al. [11] or to Hollister, 2016 [12] for details.

A total score is calculated by adding the four different factors, weighted as Peinado et al. specified (see the equation below). Peinado et al. weighted the events as the highest and scenarios as the lowest. This total score represents how far apart two separate stories are from each other, and the linear difference between the two different stories. This linear difference also demonstrated the novelty between different stories. For example, if the same story was told using different words but used the same elements, Peinado et al.'s equation would show a low linear difference. If the linear difference is high, then the stories are not similar in the essential elements.

$$Novelty = w_1 \times Events + w_2 \times Character + w_3 \times Props + w_4 \times Scenarios$$

In the equation above, all the variables that begin with *w* are the weights. No range was given for these weights – only a guidance of their values relative to each other. Each of the other items is a factor calculated according to the four equations shown above.

Peinado et al. only provided a range for the values and no guidance as to what and how he actually calculated the values. Peinado et al. apparently believed that it would be best if the researcher who used his novelty test scoring was able to develop their own weighted systems based on their own research. With this information, we performed several tests on the weighted values for this test. The first test that was conducted used simple weighting values where the lowest number began with 1 and the highest value was 4. This produced a maximum possible score of 219. A second weighted score was used with the lowest rated score was 1.1 and increased by .1 which yielded a maximum score of 36.34 points. After more experimentation, a weighted value system was used that was based with a possible maximum score of 100. Knowing that the events and characters are both important to the novelty of a story, the weights for these two events were set similarly. Likewise, the props and scenarios are both weighed similarly.

The differences between main and secondary events and between main and secondary characters were also weighted similarly, as a secondary character or event will add novelty to the story. The weights provided through experimentation are provided below for the four element equations described above. Although not quite the 100 points that were sought – the maximum number of points to be obtained are 99.96. This was close enough to 100 that one would not have to perform extra calculations to know if there was a high degree of novelty to the story.

$$Event = 1.1 \times (1.1 \times me_{new} + 1.2 \times me_{cho} + 1.3 \times me_{sim} + 1.4 \times me_{ch}) + 1 \times (1.1 \times se_{new} + 1.2 \times se_{cho} + 1.3 \times se_{sim} + 1.4 \times se_{ch})$$

$$Character = 1.1 \times (1.1 \times mc_{new} + 1.2 \times mc_{cho} + 1.3 \times mc_{sim} + 1.4 \times mc_{ch}) + 1 \times (1.1 \times sc_{new} + 1.2 \times sc_{cho} + 1.3 \times sc_{sim} + 1.4 \times sc_{ch})$$

$$Props = 1.1 \times props_{new} + 1 \times props_{changed}$$

$$Scenarios = 1.1 \times scen_{new} + 1 \times scen_{changed}$$

$$Novelty = 4.5 \times Events + 4.4 \times Character + 1.6 \times Props + 1.5 \times Scenarios$$

5.2. *Novelty Benchmark Calculations*

Using the four factor calculations as defined by Peinado et al., and the weighted scores selected experimentally (see [12] for details), an independent reviewer calculated the linear distance between the plots in Peinado et al.'s benchmark stories. The total novelty score computed was 50.14. This number represents the linear distance between two stories that have no commonality and would be considered novel in content. A novelty score of 50.14 is used as the benchmark that demonstrates that two stories are dissimilar and contains significant variance between them. These two stories provided by Peinado et al. are considered the control stories for our test, and the novelty value of 50.14 to be the benchmark value. With these values in hand, we then assessed a pair of stories generated through CCM and calculated their novelty vs. one another. We then compared that number against the Peinado et al. benchmark of 50.14. This is described in the following sections.

5.3. *CCM Test Specifics*

The objective of the novelty evaluation of stories generated through CCM was to determine whether two stories generated with the same user-defined inputs would be different from each other. The medieval genre was chosen for our test by an independent evaluator specifically selected to perform the calculations to ensure objectivity. The short story format was likewise selected to allow for consistency in comparing the novelty of these two stories with the novelty of Peinado et al.'s control stories. This would permit a valid comparison of our results to Peinado et al.'s.

As there can be diversity among arbitrary pairs of stories, the evaluations were performed on 30 different pairs of generated stories. Once the two stories were generated, each was saved to a text file to allow the independent evaluator to assess the stories in the same manner as were Peinado et al.'s control stories. The independent evaluator utilized the same process on the pairs of stories generated with the CCM process that was used in calculating the benchmark novelty in the control stories. A total point score for novelty between each of the 30 pairs of generated stories was calculated. The independent evaluator assessed all the stories in one day to maximize consistency of testing.

5.4. *Novelty Testing Results*

As per the previous discussion, the baseline novelty score for the control stories that Peinado et al. presented was 50.14. This score was designated as a reasonable score for novelty difference between two stories, as it reflects the difference between a Cinderella-like story and a randomly generated story in their research. These two stories in Peinado et al.'s research did not contain any similar events, characters, plots, or scenarios which made them totally novel to one another.

Average Novelty Score	44.26
Median Novelty Score	44.74
Standard Deviation of Novelty Score	6.88
Variance of Novelty Score	47.23

Table 1. *Novelty Scores Computed Across the 30 Runs*

A summary of the results of the novelty scores computed across the 30 runs can be seen in Table 1. These results had a great deal of variation. The average score was a 44.26 with a standard deviation of 6.88. This indicates that the control score, computed by the evaluator and considered by the evaluator to be a good benchmark to indicate novelty, is within one standard deviation of the average score. These scores are considered to be novel based on the standard deviation of the two-tailed t-*test* on the .05 level of significance that was conducted on the data. The data for all 30 runs can be found in [12] under its Appendix C. The average score of 44.26 calculates to be 44.27% of the possible maximum score of 99.96. This is indicative of the stories being labeled as novel as 44% is close to the 50% level, which was calculated by our independent evaluator for Peinado et al.'s control stories <u>using our weights</u>.

These results are more significant when one considers that the comparisons we made were not between a story created by CCM and a totally random story, as was the case in Peinado et al.'s benchmark runs. Rather, it was between two stories created by CCM of the same genre in different runs.

5.5. *Assessment of Knowledge Base Changes Required to Implement a Different Story Genre*

The cost of authorship was one concern relevant to the CCM process. In other words, how much human effort was necessary to define the knowledge required by CCM to generate its stories? This was the objective of the assessment described in this section.

As part of this assessment, new knowledge was added to a knowledge base used by CCM in an attempt to allow it to create stories in different genres (but still of the quest style). Two different story genres were utilized and the time needed to modify the knowledge base was measured. The results (time necessary to make these changes) are noted and reported in this section.

In the typical story developed through CCM, there is a hero pursuing a quest for a physical item such as "the stone of power". For this test, the needed information was added into the knowledge base to determine whether a story regarding a non-physical item such as a graduate student's quest for a Ph.D. or a candidates' quest to get elected president could be created. For example, in story #1 (the graduate students' quest for a Ph.D.), the system had to create a new main character, a new item to seek in the quest (called a Ph.D.) and added new problems that the hero needed to solve on his way to obtain a Ph.D. such as the advisor, the dissertation committee, etc. In story #2 a new main character was created to seek a new position (president), added new problems such as facing debates and winning the election. This test sought to determine the necessary effort by a (human) programmer/author to change the knowledge base in such a way as to create such very different stories.

Using selected scenarios the possible events, people, antagonists and locations that the hero could encounter were added into the knowledge base, complete with the proper genre. Each story was generated independently after the proper knowledge was added. These changes to the knowledge base were made by the first author, who was the main developer of CCM; so, there is an a priori familiarity with the processes and systems involved in the test, thus making this test somewhat optimistically biased.

Adding the basic knowledge to be able to generate the new story took the expert approximately 50 clock hours per story genre. However, there is a steep learning curve to knowing how to make these modifications and adjustments. We estimate that someone with little or no experience with the processes involved would need twice as long to accomplish the same task, or an estimated 100 hours per story type. These modifications and adjustments included how to add a new main character, how to add events and new locations in the story plot.

The full stories created by the CCM process in this test can be found in [12].

5.6. *Novelty of the Shifted Genres*

The objective of this assessment was to use the Peinado et al. [11] novelty calculations discussed above to assess the linear distance between the two shifted genre stories created above (the Ph.D. student and presidential candidate story) and one "legacy" generated story (e.g., a hero searching for a stone of power in a medieval genre). It was expected that the novelty calculation would be in the same range as was obtained in the original calculations described in the section on novelty testing above. The same objective reviewer who calculated the novelty of the stories in the novelty evaluation section was given the "shifted" stories above and performed the same calculations as completed prior. As reported above in the novelty testing section, the maximum score that could be obtained was 99.96 and the benchmark stories came to 50.14 as a basic indication of novelty.

	Event	Characters	Props	Scenarios	Novelty
Graduate Student's Quest for a Ph.D. to Legacy Story	5.25	51.4	1	1.1	49.49
Presidential Candidate Story to Legacy Story	5.14	5.25	1.1	1.1	49.64
Graduate Student's Quest for a Ph.D. to Presidential Candidate Story	5.36	5.15	1	1	49.88

Table 2. *Novelty Scores of Shifted Stories*

A novelty of 49.49 (49.5%) was calculated as the novelty between the legacy (stone of power) story and a graduate student's quest for a Ph.D. The generated story may be seen in [12]. A novelty rating of 49.64 (49.7%) was calculated for the presidential candidate's story. Another calculation that compared the two stories to each other (the Ph.D. student vs. the presidential candidate) had a novelty rating of 49.9% (49.88). A table containing the novelty scores can be seen in Table 2.

All three of these scores are near the 50% rate that was considered to be the benchmark for novelty in the novelty evaluation section above. The scores are all within the same range of novelty as calculated in the novelty section. This further indicates the ability of CCM to create novel stories.

6. Summary and Discussion

This paper has introduced a new method of story generation utilizing a contextual approach that allows the listener to request changes and hear stories that incorporate those changes. This is a practical and innovative approach that can enhance the entertainment and novelty value of the story being generated. Because of page limitations, the extensive testing performed on CCM cannot be totally displayed here. We refer the reader to [12] for a complete discussion of testing and the results obtained.

Although this method has only been applied in this paper to the generation and telling of stories, it could potentially be applicable to other areas. CCM could provide narratives for computer games or help create scenarios for simulation-based tactical training, such as for military operations and first responder training. These will be pursued in future research.

7. References

[1]Gonzalez A.J., "Tactical Reasoning through Context-Based Reasoning", in Brezillon, P. and Gonzalez, A.J. (eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York, NY: Springer Science + Business Media, 2014.

[2] Turner R.M., Context-mediated behavior for AI applications. In: Mira J, del Pobil AP, Ali M (eds) *Methodology and Tools in Knowledge-Based Systems: 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Volume I. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 538-545, 1998.

[3] Brézillon P., "Representation of Procedures and Practices in Contextual Graphs". *The Knowledge Engineering Review*, 18(2), pp. 147-174, 2004.

[4] Dey A.K., "Understanding and Using Context" *Personal and Ubiquitous Computing*, 5(1), pp. 4-7, 2001.

[5] Parker J.E., Hollister, D.L., "The Cognitive Science Basis for Context." in Brezillon and Gonzalez (eds.) Context in Computing: A Cross-disciplinary Approach for Modeling the Real World, New York: Springer, pp. 205-219, 2001.

[6] Booker C., The Seven Basic Plots: Why We Tell Stories. Bloomsbury Academic, 2004.

[7] Stoker B., *Dracula*, 1925.

[8] Melville H., *Moby Dick*: new American Library.Verne, J. (1887). *Twenty Thousand Leagues Under the Sea*: Butler Brothers, 1892.

[9] Verne J., *Twenty Thousand Leagues Under the Sea*: Butler Brothers, 1887.

[10] Wells H.G., *The War of the Worlds*: Watermill Press, 1898.

[11] Peinado F., Francisco V., Hervás R., Gervás P., Assessing the Novelty of Computer-Generated Narratives Using Empirical Metrics. *Minds and Machines, 20*(4), pp. 565-588, 2010.

[12] Hollister J., A Contextual Approach to Real Time Interactive Narrative Generation. Doctoral dissertation, University of Central Florida, 2016.

[13] Gonzalez A.J., Saeki, S., “Using Context Competition to Model Tactical Human Behavior in a Simulation”, Proceedings of the CONTEXT-01 Conference, Dundee Scotland, 2001.