Applying Qualitative Research Methods to Narrative Knowledge Engineering*

Brian O'Neill[1] and Mark Riedl[2]

1 Department of Computer Science and Information Technology, Western New England University
1215 Wilbraham Rd., Springfield, MA, USA
brian.oneill@wne.edu

2 School of Interactive Computing, Georgia Institute of Technology
85 Fifth St., Atlanta, GA, USA
riedl@cc.gatech.edu

Abstract

We propose a methodology for knowledge engineering for narrative intelligence systems, based on techniques used to elicit themes in qualitative methods research. Our methodology uses coding techniques to identify actions in natural language corpora, and uses these actions to create planning operators and procedural knowledge, such as scripts. In an iterative process, coders create a taxonomy of codes relevant to the corpus, and apply those codes to each element of that corpus. These codes can then be combined into operators or other narrative knowledge structures. We also describe the use of this methodology in the context of Dramatis, a narrative intelligence system that required STRIPS operators and scripts in order to calculate human suspense responses to stories.

1998 ACM Subject Classification I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases narrative intelligence, qualitative methods, coding, knowledge engineering

Digital Object Identifier 10.4230/OASIcs.CMN.2014.139

1 Introduction

Narrative intelligence includes the ability to generate narratives, explain experiences in narrative terms, and understand and make inferences about narratives. Computational narrative intelligence tasks, such as story generation and story understanding are knowledge-intensive processes. A system would have to know everything that a human would be expected to know about the story domain, and that knowledge could be extensive. For simple domains, such as going to a fast-food restaurant, a narrative intelligence system would need scripts describing the relationship between the customer and the staff, how that interaction changes when using a drive-thru, as well as an understanding of the actions available in such a scenario. As the domain becomes more complicated, the space of required knowledge grows. Compare the fast-food restaurant domain to the knowledge necessary for *James Bond* movies. For the latter domain, it would be necessary to encode the types of problems that

* This material is based upon work supported by the National Science Foundation under Grant No. 1002748. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

spies tend to face, the actions that a spy might take to overcome those problems, as well as knowing how to incorporate the gadgets that Bond uses.

Creative narrative intelligence systems, such as story generation and story understanding systems, are knowledge-intensive. Story generation systems typically use a planning approach or a case-based reasoning approach. Planning approaches require knowledge in the form of domain specification (e.g. STRIPS or PDDL), while case-based reasoning approaches require a library of cases. Story understanding systems also frequently use case-based reasoning. When a knowledge-intensive narrative intelligence system demonstrates creativity, it is not clear where credit for that creativity should properly be assigned. Is the creativity, such as the generated story, the result of a good algorithm? Or is it the result of a well-crafted domain? If it is the latter, then the creativity should be ascribed not to the system, but to the designer of the domain.

We propose a methodology for converting a natural language corpus into a domain specification for narrative intelligence systems. Current approaches to converting corpora into domain specifications rely too heavily on the knowledge engineer. By leveraging approaches from qualitative methods research such as ethnography—methods specifically intended to elicit information from texts without being affected by researcher bias—we can construct a domain while limiting the influence of the designer. When domains are constructed using this methodology, it is possible to make stronger claims about the origins of the resulting system's creativity. Because the domain is not the result of too careful crafting by the designers of the system, we can conclude that system creativity is the result of the algorithm rather than knowledge engineering.

In this paper, we present a methodology for extracting narrative knowledge from natural language data and its conversion to a domain usable by a narrative intelligence system. This methodology uses formalized *coding* procedures from qualitative methods research to identify actions and other narrative knowledge in the corpus. Over multiple iterations, coders identify common actions and themes in the corpus. These common themes are used to form a taxonomy of codes which can then be applied by multiple coders to the entire corpus. These codes can then be used to generate narrative knowledge structures, such as scripts and operators. We also describe the application of this methodology to create STRIPS operators and scripts for Dramatis [18], a narrative intelligence system that models human suspense responses to stories.

2 Related Work

2.1 Knowledge Acquisition

A large number of narrative intelligence systems require background knowledge, such as scripts, plans, or other formalisms [5, 26, 15, 9]. In many cases, this knowledge is formed manually. Manual generation leaves the knowledge base prone to the biases of the domain engineers. As a result, artificial intelligence researchers have attempted to automate the acquisition of procedural knowledge (e.g. scripts) from natural language corpora. The Say Anything interactive storytelling system uses sentences collected from a corpus of blog posts [25]. However the resulting stories still needed assistance from human users to ensure that the stories were coherent. Chambers and Jurafsky [2] learn "narrative event chains," which are single-character script-like sequences of events. They analyzed the Gigaword corpus to learn the significant events of a sequence and used machine learning approaches to generate a partial-ordering of these events. Fujiki et al. [8] analyzed Japanese newspaper articles in order to acquire scripts about murder cases. Kasch and Oates [10] collected a corpus of web documents pertaining to a target subject. Once the field of relevant documents has been narrowed, their procedure locates pairs of events based on argument co-reference in order to create script-like knowledge structures. In each of these examples, researchers had access to a sizable corpus of relevant texts, such as newspaper articles about murders.

Depending on the domain in question, the existence of a large useful corpus is not guaranteed. When such corpora are not available, it is possible to use humans to generate specialized corpora. *Human computation* refers to systems that organize people to carry out computational processes, such as tasks that machines cannot typically carry out effectively [13]. A growing form of human computation is *crowdsourcing*. This approach attempts to use "the wisdom of the crowd," where it is believed that the knowledge of a large number of people is superior to that of a single person [13]. In crowdsourcing, the human computation task is distributed to a large pool of people. Frameworks such as Amazon Mechanical Turk (AMT) have been developed as a means of distributing human computation tasks to large numbers of workers, evaluating the quality of the work, and paying them for their brief participation. As human computation and crowdsourcing have grown, researchers have attempted to delegate the acquisition and aggregation of procedural knowledge to large collections of people rather than to automated processes. Boujarwah et al. [1] implemented a process for acquiring scripts from AMT workers. They later used other AMT workers to classify and evaluate the quality of the responses received in the initial script collection phase. Li et al. [14] asked AMT workers to provide the typical events of particular stories, such as dates and bank robberies. In this data collection, workers were given specific instructions about the nature of their responses, such as using simple sentences and only one verb per sentence. Using the crowd-acquired corpora, they automated the learning of script-like structures called plot graphs. However, this approach does not create new stories from the actions in the domain. Instead, it repeats sequences of events that have been provided by the AMT workers. ScenarioGen generated scenarios for serious games using a procedure that combines crowdsourcing with automation [23]. This approach collects scenarios from the crowd, as well as soliciting possible replacements for events in order to create new scenarios. ScenarioGen used satisfiability solvers and K-nearest neighbor techniques to identify when scenarios may require substitute events. Finally, the crowd is utilized again to evaluate the resulting scenarios. While each of these strategies are effective for acquiring narrative knowledge, each comes with a cost. Using AMT workers in three phases—initial collection, classification, and quality control—is a costly proposition when paying workers in multiple domains. While automation reduces the cost of crowd workers, there remains a time cost in ensuring that the learning algorithms are structuring the data appropriately.

2.2 Qualitative Methods

Coding is a qualitative research method used to elicit concepts, theories, or key phrases from natural language data, such as interview transcripts, journals, videos, and other subjective data [20]. It is a common process in fields that heavily utilize qualitative data, such as learning sciences and human-computer interaction. In some cases, coding is one step of a larger approach to qualitative research, such as grounded theory or thematic analysis. A *code* is a word or phrase that summarizes the key details of some aspect of the media being coded. When considering interview data, a code may be applied to a paragraph, or multiple codes may be applied to a single sentence, depending on the particular coding technique being used and the contents of the data. The coding process allows for the identification of similarly themed data when codes are analyzed. Coding is often an iterative process wherein codes are refined as researchers become more familiar with the data and the common themes, or as they attempt to form distinct categories from the data.

Coding is designed for use on the same types of natural language corpora from which artificial intelligence researchers have been attempting to extract procedural knowledge for years. When applied to this context, coding serves as a formalized process for identifying various types of narrative background knowledge, such as actions or event chains, that are implicit within a variety of natural language texts.

3 Methodology

We introduce the following methodology for using coding to convert natural language corpora into knowledge structures for narrative intelligence systems. The description of the methodology is intended to be agnostic as to the source of the corpus as well as the particular representations of the desired knowledge structures. In order to allow for a wide variety of source materials and intended knowledge structures, some decisions in this process are left to the researchers. Some may find it useful to adapt or alter this methodology to better meet the goals of their particular narrative intelligence system. The remainder of this section describes the methodology broadly, while the following section describes the use of this methodology with a particular system.

3.1 Creating a Corpus

This methodology requires a natural language corpus as a source of knowledge for the system under development. The origin of this corpus is not relevant to the procedure and is ultimately dependent on the system in question. Many of the procedures described in related work begin with identifying corpora, each of which would be suitable for this methodology. Surveys, crowdsourced materials, blog posts, game traces, or any other natural language source are applicable to this approach, where the best choice depends on the type of knowledge the researchers wish to encode.

3.2 Coding the Corpus

The corpus is coded in a four-stage process adapted from qualitative methods processes used to parse interview transcripts and ethnographic data, among other tasks. Each individual item (e.g., web page, text sample, or survey response) in the corpus is treated as though it were an interview transcript. For the purposes of this methodology, we will refer to individual sentences or survey answers as *entries*. An *entry* should be the smallest unit of the corpus from which actions will be extracted.

The four stages can be briefly described as follows:

1. Code the corpus by identifying actions, as well as potentially problematic entries within the corpus.
2. Combine actions and problems into broader categories, defining guidelines for what attributes indicate that an entry belongs in a particular category.
3. Multiple coders independently code a subset of the corpus, using the coding guidelines established in the previous phase. Repeat this step until a sufficiently high level of inter-coder agreement is achieved.
4. A single coder from the previous stage codes the remainder of the corpus according to the same guidelines.

The first phase is based on a coding technique known as *Initial Coding* [20, 3]. Initial Coding is a "first cycle" coding method, where researchers produce tentative codes that will later be refined before overall analysis. This process also uses aspects of *In Vivo Coding*

[3, 24], which guides coders to create codes based on the actual words of the corpus. During this phase, a single person codes each entry of the corpus. Entries containing actions should be coded with the verb in the sentence. For example, the entry "The spy orders a drink." should be coded as the action `order`. If there is reason to believe that an entry is not appropriate for conversion to a target knowledge structure, then the entry should be coded with a brief explanation of the problem. Reasons for exclusion depend on the coding task and the system in question and may not apply to all domain engineering tasks. Potential reasons for exclusion could be that the response ignored the survey prompt or presented irrelevant setting details rather than actions (e.g., the entry "It was a beautiful day." could be coded as `setting`).

The second stage uses a process known as *Focused Coding* [3]. This technique is a common "second cycle" coding method that is frequently applied after Initial Coding. The goal of Focused Coding is to identify patterns and categorize the codes created during the first cycle. During this phase, a domain engineer combines the non-action codes from the first stage into a taxonomy of codes that represents the space of possible rejection reasons. Researchers should also create a general code for acceptable entries. Depending on the corpus or the desired knowledge structures, it may be useful to create several codes for entries that represent acceptable actions. For example, it may be useful to have a code indicating that the entry describes multiple actions. The codes created in this phase will be used in later phases. For each code in the new taxonomy, the domain engineer should create guidelines indicating when an entry should be coded as part of this set rather than a different set. The exact number of codes created in this phase, and the breadth of those codes, is ultimately dependent on the corpus and the knowledge domain.

While we describe these first two phases in terms of Initial and Focused Coding, one could also view these phases as a form of *Provisional Coding* [20, 16]. In this technique, researchers establish codes prior to data analysis based on prior experience, related work, and their own expectations and hypotheses. The resulting set of codes can later be modified if observations reveal the need for new codes or a finer level of granularity.

In the third stage of this process, multiple coders (possibly including the original coder from the previous phases) independently code a subset of the corpus using the codes and guidelines created in the previous phase. A sufficiently high level of agreement and inter-rater reliability would indicate that one of the coders could continue to code the remainder of the corpus alone in the final stage with a relatively low risk of error. Using multiple coders in this phase reduces the risk of error, while increasing the confidence in the codes applied to each entry. The corpus subset should represent approximately 20 percent of the full corpus. If multiple prompts were used to develop the corpus, or if there are clear categories of samples within the corpus, each prompt or category should be proportionally represented in the corpus subset. In this phase, when coders apply the code (or one of the codes) for acceptable actions to a particular entry, the code should also specify the action represented in the entry. Taking this step mimics the In Vivo coding technique used in the first phase, as coders should attempt to use the words that were present in the entry. Thus, the entry "The spy orders a drink." would be coded as `Action/order`. Entries that receive rejection codes do not need to incorporate additional information.

After this subset has been coded, calculate the inter-rater reliability amongst the coders. For the purposes of this paper, we use Cohen's κ, though this metric is one of several useful inter-rater reliability metrics and using this one is not essential to the process. Using Cohen's κ, scores greater than 0.6 are typically considered "good" inter-rater reliability, while values greater than 0.8 represent "excellent" agreement [12, 4, 7, 16]. Before starting this phase, researchers should determine what level of inter-rater reliability is sufficient for their knowledge engineering problem and corpus. Statisticians recognize that this threshold is arbitrary, ultimately depending on the task and importance of agreement [11, 20]. Referring to his own *alpha* measure of inter-rater reliability, Krippendorff stated that a threshold of 0.667 could be applicable under certain circumstances [11].

Defining agreement for non-action codes is simple. However, it may be challenging to determine what constitutes agreement in the case of action codes. At a high level, coders may agree that an entry is an action. At a more fine-grained level, agreement may depend on the coders applying similar or identical words to the entry. Using In Vivo coding helps ensure that coders agree on the described action by using the exact words in the corpus.

After any iteration of this phase where inter-rater reliability did not meet the target threshold, the coders should gather and discuss the codes and guidelines. The coders may revise codes or guidelines, or add new codes, in order to improve the level of agreement in the subsequent iteration. Once these revisions are made, the coders should independently return to the corpus subset, coding according to the revised guidelines. These iterations continue until the intended level of inter-rater reliability is achieved. Iterative processes such as this phase are common in qualitative methods such as grounded theory [3].

Once the coders have reached a sufficient level of agreement, a single coder from that group may begin the fourth phase. In this phase, the individual coder applies the most recent revision of the coding guidelines to the remainder of the corpus. Additionally, the coder should resolve any remaining disagreements in codes from the previous phase. This resolution may come from unilateral decision making or through consensus agreement of the several coders. At the end of this phase, each entry in the corpus has a single tag as one of the following:

- An action, and the action that is indicated by the entry. This may be further extended if the coding taxonomy used sub-categories for actions.
- A candidate for rejection, based on the particular code from the taxonomy generated during Focused Coding and revised in the iterative process during the third phase.

3.3 Generating Knowledge Structures

Having fully coded the corpus, it is now possible to convert these codes into the desired knowledge structures. The specific conversion processes depend on the representations used by the narrative intelligence system. For example, converting to Schankian scripts [21] will require a different process than converting to the event chains used by Chambers and Jurafsky [2], or the plot graphs used by Li et al. [14]

In general, each entry in the corpus has now been coded, either as an action or with a reason to exclude the entry. These actions indicate a set of operators in the corpus. Depending on the system or representation, domain engineers may wish to further narrow this set of actions by combining like actions. For example, entries coded as `Action/walk` or `Action/drive` could be combined into the more generic `go` operator, so long as it is not necessary to distinguish between the two original actions in this particular domain. If the engineering task requires some piece of data that was not part of the coding process (e.g., perhaps the process excluded causal information, but the operators being engineered need preconditions), then the domain engineers may have to create this information themselves or infer the information from the original corpus materials.

Given that each entry in the corpus is now coded as an action, the larger items of the corpus (survey responses, web pages, articles, etc.) now contain sequences of actions, which could be converted to narrative structures akin to scripts. As with operator generation, the precise details depend on the nature of the representation. Similarly, domain engineers will need to determine for themselves how to handle rejection codes that appear mid-sequence. Leaving these items out could damage the coherence of a script, but including bad information could be detrimental to the domain.

In any case, the formalisms of the desired knowledge structures will define much of the conversion process. These formalisms should be kept in mind throughout the coding process, particularly during Initial Coding and Focused Coding.

4 Method in Practice

In this section, we describe an implementation of the process described above, as well as some of the decisions that were made as a result of this context. We describe the qualitative knowledge engineering methodology in the context of a narrative intelligence system that uses scripts and plans to calculate the suspense level of stories.

4.1 Dramatis

Our knowledge engineering problem was related to Dramatis, a computational model of suspense based on psychological and narratological understandings of the suspense phenomenon [18]. In order to recognize suspense in the stories it read, Dramatis required a library of actions that could occur in the domains of those stories. These actions are represented as STRIPS operators [6], which are used to plan solutions to possible negative consequences faced by the protagonist. The operator library should include actions that are not present in the story, so that the model could produce alternate solutions to the protagonist's problems. Additionally, Dramatis required script-like structures that indicate typical sequences of events in the story domains. Dramatis uses these scripts to predict possible future events which may affect the level of suspense in the story. In order to evaluate the Dramatis model, we needed to collect these planning operators and scripts.

The planning operators used by Dramatis are typical STRIPS operators, made up of the action name, a set of parameters, and two sets of propositions representing the action's preconditions and effects. With this methodology, our goal was to collect the actions represented by these operators. This acquisition process was not expected to give the parameters, preconditions, or effects for the operators. Parameters would be determined after the fact, based on the context of the actions in the original corpus. The causal propositions were engineered afterwards, so that it would be clear which elements of the domain were necessary to represent and which were irrelevant. However, by collecting operators from an outside corpus, we were able to ensure that a wide variety of relevant operators were included, rather than focusing only on those that occurred to the knowledge engineers.

The scripts used by Dramatis are graphs where nodes represent events and edges represent either a temporal ordering relationship or a causal relationship between those events. Events are represented in the script nodes by the corresponding STRIPS operator. Thus, by collecting actions and converting them to STRIPS operators, we are able to collect the nodes of the script graphs. Because the corpus we used contained sequential event information, we also collected the temporal links for the scripts. The causal links were directly related to the STRIPS operators. Because operator preconditions and effects were authored after the fact, causal links for the script graphs could not be added until the STRIPS operators were complete. Figure 1 shows a fragment of a script for a spy story domain created using this methodology, where solid lines indicate temporal links and dashed lines indicate causal links.

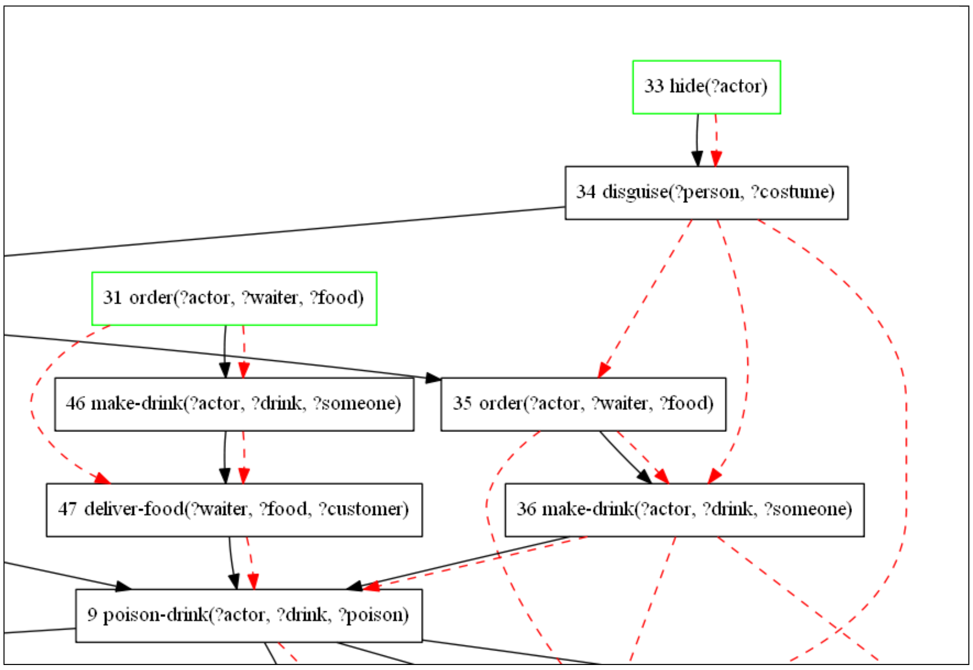
33 hide(?actor)
34 disguise(?person, ?costume)
31 order(?actor, ?waiter, ?food)
46 make-drink(?actor, ?drink, ?someone)
35 order(?actor, ?waiter, ?food)
47 deliver-food(?waiter, ?food, ?customer)
36 make-drink(?actor, ?drink, ?someone)
9 poison-drink(?actor, ?drink, ?poison)

Figure 1 A fragment of a script in the Spy domain.

4.2 Corpus Creation

In order to generate the operators and scripts, we first needed a natural language corpus for the domains which would be used to test Dramatis. These domains were adapted from suspenseful scenes in popular films. The scenes selected were:

- From the film, *Casino Royale*, the scene where James Bond is poisoned at the poker table and attempts to cure himself.
- From Alfred Hitchcock's film, *Rear Window*, the scene where Lisa breaks into Thorwald's apartment to find evidence that Thorwald murdered his wife.

Though we identified these two scenes, it was insufficient to simply use the actions within the scenes as the operators and scripts. While those actions should be included in the operator library, it was necessary for Dramatis to be able to consider the same space of actions that were likely to be considered by human viewers. Using only the actions from the source material would provide the solution, but it would not accurately describe the space of actions available to the characters and the viewers planning on the characters' behalf.

Based on these scenes, we developed three survey prompts based on the crowdsourcing tasks used by Boujarwah et al. and Li et al. [1, 14]. Each prompt described the beginning and end of one of the scenes. Respondents were instructed to list the steps that occurred in the story between these two points. Two prompts were created for the *Casino Royale* example. The *Spy 1* prompt asked participants to describe how a spy could go from being in a bar to being poisoned. The *Spy 2* prompt asked how a spy could go from being poisoned to no longer being poisoned. The *Rear Window* prompt described a scene where two people suspected their neighbor of murder. One of these people was on their way to the neighbor's apartment in search of evidence. Participants were asked to describe the events from entering the neighbor's apartment to being caught intruding by the suspected murderer. Each prompt was written to avoid reference to its source material. The *James Bond* prompts refer to a

Table 1 Knowledge Acquisition Study prompts.

Spy 1	Start: A spy is at a bar or restaurant. Finish: The spy drinks from a drink poisoned by the villain.
Spy 2	Start: A spy is at a bar or restaurant. The spy just drank from a drink that was poisoned by the villain. Finish: The spy is no longer poisoned.
Rear Window	Start: A man (Tom) and a woman (Erin) suspect their neighbor of committing murder. Tom cannot leave the apartment, but Erin has just left the apartment to sneak into their neighbor's apartment to find proof. Tom and Erin have an agreed upon signal for if the neighbor is on his way home. Finish: The neighbor catches Erin in his apartment.

Table 2 Statistics of Knowledge Acquisition Study responses.

Prompt	**No. Responses**	**Total No. of Entries**	**Median Entries**	**Mean Entries (SD)**
Spy 1	18	131	7	7.28 (3.78)
Spy 2	24	168	5	7.00 (4.79)
Rear Window	18	198	9.5	11.00 (5.11)

generic spy, while character names were changed in the *Rear Window* prompt. Table 1 shows the specific prompts given to participants.

Each prompt was placed in a Google web survey, with 20 numbered blank text fields. The instructions asked participants to describe the events between the prescribed start and end points in the fields provided in order. Additionally, the instructions specified that responses should focus on events or actions rather than setting. Finally, the instructions noted that participants were not required to use all 20 text fields. Prospective respondents were directed to a webpage where all three surveys had been embedded in a random order. Respondents were recruited using institution mailing lists and social media. Table 2 shows the response rates, as well as the average number of text fields used in each response.

4.3 Coding Process

One of the authors of this paper coded the survey responses according to the Initial Coding procedure described in Section 3.2. In the Focused Coding phase, the non-actions codes were reduced to a taxonomy of eleven codes that represented the space of possible reasons for exclusion. Additionally, action codes were divided into two codes: a code for entries representing single actions, and a code for entries representing multiple actions. These thirteen codes were used in the third phase of the coding procedure.

Table 3 shows the thirteen codes and the guidelines used for applying these codes. The reasons for exclusion varied. The most pressing reason was signified by the `Prompt Failure` code, which indicated that the end state of the respondent's story did not match the end state requested by the prompt. Similarly, we coded entries for exclusion when they described the setting rather than actions (`State` code), or provided multiple possibilities for actions without committing to a single action (`Vague` code). We also excluded entries that took an audience point-of-view by referring to discourse-level details, such as events being presented in flashbacks (`Presentation` code). Other exclusion reasons included characters taking

Table 3 Coding Guide for Knowledge Acquisition Responses.

Code Type	Shorthand	Description
Single action	[Specify action]	Applies when the entry describes a single action/operator. Provide the operator in the response.
Multiple actions	[Specify actions]	Applies when the entry describes multiple actions/operators.
Prompt Failure	PROMPT	Applies when the end state of the response does not match the state instructed by the prompt.
Attention	ATTN	Applies when an entry deals with what a character is paying attention to or noticing.
Dialogue Action	DLG	Applies when an entry deals with what a character said. Does not apply when the entry just says two characters talked.
State	STATE	Applies when an entry provides state information but no action.
Thoughts	THGT	Applies when an entry deals with what a character is thinking or thinking about.
Inaction	INACT	Applies when an entry describes a character explicitly not taking an action.
Presentation	PRES	Applies when an entry describes audience point-of-view or sjuzet details.
Incomplete Actions	INC	Applies when a character begins performing an action or task but does not complete it.
Continuation	CONT	Applies when an entry is a continuation of the previous entry, or of the action described in the previous entry.
Continuing Failure	CF	Applies when an entry represents multiple attempts to do something with repeated failure and/or no expectation of immediate success and/or waiting for something to happen.
Vague	VAGUE	Applies when an entry says something happens, but not how; or when an entry provides multiple options for what might have happened.

actions that required modeling their inner state (`Attention` and `Thought` codes) or actions that failed or were repeated over the course of several entries.

The third phase of coding was conducted by the same author as the Initial Coding and a partner. For this phase, we randomly selected five responses from each prompt for the subset, amounting to 23% of the survey responses. During this phase, the two coders agreed on 76.3% of codes (Cohen's $\kappa = 0.64$). Additionally, every time that both coders marked entries as actions, there was semantic agreement about what action was represented by that entry. When codes were reduced to a simple Accept/Reject question (where Accept is a single action or multiple actions, and Reject is any of the eleven non-action codes), the coders agreed on 83.9% of codes ($\kappa = 0.67$). Prior to this phase, we agreed that "good" inter-rater reliability was sufficient. Therefore, only one iteration was necessary during this stage.

In the final phase, the same author coded the remainder of the survey responses according to the same guidelines shown in Table 3. Any coding disagreements from the previous phase were resolved through consensus, though the only remaining disagreements came from entries being coded as actions by one person and given non-action codes by the other coder. At

Table 4 Sample Survey Response with Initial and Final Codes.

Response	Initial Code	Final Code
A man (Tom) and a woman (Erin) suspect their neighbor of committing murder. Tom cannot leave the apartment, but Erin has just left the apartment to sneak into their neighbor’s apartment to find proof. Tom and Erin have an agreed upon signal for if the neighbor is on his way home.	Restatement of prompt	STATE - State information
Erin discovers a red herring.	Action - discover	Single action - discover
Erin becomes afraid of a noise.	Emotion	THGT - Thoughts
Erin realizes the noise was something innocent.	Realization	TGHT - Thoughts
Erin finds a clue.	Action - find	Single action - find
Erin goes where she cannot see Tom’s signal.	Action - go	Single action - go
Erin finds gruesome evidence.	Action - find	Single action - go
Erin hears the neighbor arrive home.	Passive, hearing things	ATTN - Attention
Erin hides.	Action - hide	Single action - hide
Erin continues to hide as the neighbor moves.	Continuing action, action - move	CONT - Continuation
The neighbor catches Erin in his apartment.	Restatement of prompt	Single action - catch

the end of the process, each entry from each survey response had been tagged as one of the following:

- A single action, and what action is indicated.
- Multiple actions, and what actions are indicated.
- A candidate for rejection, along with the specific rejection code from Table 3.

Table 4 shows one complete response to the Rear Window prompt. The middle column shows the results of the Initial Coding process, while the last column shows the codes after all phases of coding had been completed. This particular entry was coded by both coders. The only disagreement between the coders came on the third entry. One coder listed the entry with the rejection code for character thoughts, while the other coded the entry as a single action `become-afraid`. During the final phase, this disagreement was resolved through consensus, and the rejection code was ultimately selected.

4.4 Generating STRIPS Operators

Prior to converting the coded survey responses to the knowledge representations used by Dramatis, we removed any response with an entry coded as `Prompt Failure`. This code indicated that the respondent did not adhere to the prompt, typically by failing to meet the specified conditions at either the beginning or the end of the story. As a result, the entire response was not useful. Other rejection codes only affected the single entry rather than an entire survey response.

After the coding process was completed, each identified action was converted into a STRIPS operator [6]. Similar actions (e.g. “sneak” is a special case of “go”) were combined into single operators. The coding process provided the action, or the verb, for the operator. However, STRIPS operators require parameters, preconditions, and effects, none of which

```
operator:  deliver-food (?waiter ?food ?customer)
    constraints:  person(?waiter) person(?customer) edible(?food) (neq
        ?waiter ?customer)
    preconditions:  has(?waiter ?food) ordered(?customer ?food)
        waiter(?waiter)
    adds:  has(?customer ?food)
    deletes:  has(?waiter ?food) ordered(?customer ?food)
```

Figure 2 Example Planning Operator.

were immediately derivable from the survey responses during the coding process. In some cases, parameters could be inferred from the original text entry, such as parameters which pertained to the subject or direct object of an action. STRIPS operators often require additional parameters that describe details that are implied by natural language. For example, the `give` operator would require a location parameter to make sure that both characters involved are co-located. However, the single sentence describing the act of giving usually would not contain location information. Operator preconditions and effects were inferred from how the actions were used in the survey responses, rather than from the coding process. Preconditions and effects were also modified later as the operators were tested and interactions between them were observed.

Figure 2 shows an operator created from the Spy prompts. The `operator` row shows the operator name and parameters. The `constraints` and `preconditions` lines show operator constraints and preconditions, where constraints are a special subset of preconditions that establish immutable facts about the parameters in question, such as a parameter variable referring to a person. The `adds` and `deletes` lines refer to propositions that are added and deleted, respectively, from the world state as effects of the operator being completed. The full set of operators created for both the Spy and Rear Window domains can be seen in [17].

4.5 Generating Scripts

After the operators were finalized, we combined the survey responses into a script for each prompt. Each survey response represented a portion of the script, making up a path through the script graph. When entries were coded as actions, the corresponding operator was added to the scripts. Entries coded as non-actions were skipped, unless doing so affected the coherence of the story in the survey response. Typically, an existing operator was relevant to the entry despite the code. Additionally, in some cases, events were included in the script trace that had been left implicit in the original survey response (e.g., the operator `make-drink` was specified between `order` and `deliver-drink` by some participants, but not all). Figure 1 shows a portion of the script created from the Spy 1 prompt. The complete scripts for the Spy and Rear Window domains can be seen in [17].

Additional information was added to the script representation once the sequences of actions had been collected from the coded survey responses. Dramatis scripts required causality information about the events of the script. These causal links were added based on the preconditions and effects that were created for the operators. We also annotated the script so that it was clear when the same character was expected to perform several actions. These annotations were derived from the survey responses directly.

4.6 Discussion

The resulting operators and scripts were successfully used to evaluate the Dramatis model [18]. The knowledge acquisition and coding procedures led to 62 operators for the *Casino Royale* scene based on the two spy domain prompts, and 38 operators for the *Rear Window* domain. The *Casino Royale* and *Rear Window* scripts had 51 event nodes and 44 event nodes, respectively. Dramatis used these operators and scripts to find possible solutions for characters facing negative consequences, which was part of the process of calculating suspense responses. In system evaluations, we demonstrated that Dramatis produced suspense ratings that corresponded to ratings produced by human readers, in part because of how the model used this narrative knowledge [18].

It is possible that a second iteration of the third phase, using multiple coders to code a subset of the survey responses, could be beneficial to the knowledge structures used for Dramatis. While we were satisfied with the "good" Cohen's κ of 0.64, we were still distant from "excellent" agreement ($\kappa \geq 0.8$). Further iterations would provide greater confidence in the individual coding completed in the fourth phase of the process. However, it is notable that this inter-rater reliability calculation does not take into account the semantic agreement on the actions between the two coders. Rather, it only notes when both coders marked an entry as `Single action` or `Multiple actions`. Accounting for the agreement in action descriptions might increase the inter-rater reliability calculation.

It is important to recognize that the codes used for Dramatis (Table 3), while appropriate for our prompts, are not necessarily applicable to all knowledge engineering tasks. Other researchers will need to go through the same initial processes, using Focused Coding to develop their own taxonomy of codes, allowing the codes to emerge from the corpus. It is not difficult to imagine other systems that have other criteria or would want to include entries that we excluded. For example, where we excluded dialogue actions, others may want to encode such entries in their knowledge base. These decisions must be made prior to Initial Coding and depend entirely on the goals of the researchers.

5 Future Work

While we were able to successfully generate operators and procedural knowledge using this methodology, we needed to author causal knowledge by hand after the fact. Future work should focus on how to extract causal information from natural language corpora using qualitative methods. It may be possible to build on the work of Sil and Yates [22] in order to collect some of this causal information automatically. Further effort is also necessary to determine the best way to collect causal information from the crowd. While crowdsourcing has proven effective for providing sequences of events, it is not yet clear whether it is reasonable to expect untrained AMT workers or survey respondents to provide the level of causal information needed to produce STRIPS operators or other structures including causality. A number of narrative systems also consider the intentionality of its characters [19], which this methodology has not addressed. Further research will help determine how corpora can be coded in order to extract the goals and intentions of the actors described in the texts. Finally, it may be valuable to extend this process beyond natural language texts to media such as films and games. Including these other media will likely require alterations to the methodology. However, the gains for narrative researchers will be significant if they are not limited to text-only formats when using this approach.

It may be useful to evaluate this methodology by comparing it to one of the other knowledge acquisition and engineering processes discussed previously. For example, if we could evaluate the quality of the knowledge structures generated using this approach, we could compare the resulting structures to those created using the automated processes described by Li et al. [14] or other hand-tailored approaches to knowledge engineering. Evaluating the quality of a knowledge structure remains an open questions, but could perhaps be accomplished by considering how well the narrative intelligence system performs with that particular set of knowledge.

6 Conclusions

We have introduced a methodology for creating a narrative intelligence domain from natural language corpora using techniques from qualitative methods research. This technique mitigates the influence of system designers in crafting the knowledge needed by the system in question. Additionally, we demonstrated the use of this methodology in the context of Dramatis, a system that demonstrates narrative intelligence by calculating a reader's suspense response. Our methodology was used to generate STRIPS operators and scripts which Dramatis used as part of it calculations.

By limiting the role of the designer in the knowledge engineering process, designers can make stronger claims about the creativity of their systems. Using this methodology, we can assign credit for creative results to the algorithms used by narrative intelligence systems, rather than to the domain designer. Knowledge-intensive systems, such as story generators and story understanding systems, will always need knowledge that is comparable to what humans would be expected to know in the same domains. Codifying the process for converting from corpora to domain, while simultaneously mitigating the influence of the designer, will allow researchers to have greater confidence in the source of the creativity of their systems.

References

1 Fatima A. Boujarwah, Gregory D. Abowd, and Rosa I. Arriaga. Socially computed scripts to support social problem solving skills. In *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems*, pages 1987–1996, Austin, Texas, USA, 2012. ACM Press.

2 Nathanael Chambers and Dan Jurafksy. Unsupervised learning of narrative event chains. In *Proceedings of the Forty-Sixth Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 789–797. Association for Computational Linguistics, 2008.

3 Kathy Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis.* Sage Publications, Thousand Oaks, California, USA, 2006.

4 Domenic V. Cicchetti. Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology. *Psychological Assessment*, 6(4):284–290, December 1994.

5 Richard E. Cullingford. SAM and micro SAM. In Roger Schank and Christopher Riesbeck, editors, *Inside Computer Understanding.* Erlbaum, Hillsdale, NJ, 1981.

6 Richard E. Fikes and Nils J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

7 Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, November 1971.

8 Toshiaki Fujiki, Hidetsugu Nanba, and Manabu Okumura. Automatic acquisition of script knowledge from a text collection. In *Proceedings of the Tenth Conference on European*

chapter of the Association for Computational Linguistics, pages 91–94, Budapest, Hungary, 2003. Association for Computational Linguistics.

9 Pablo Gervás, Belén Díaz-Agudo, Federico Peinado, and Raquel Hervás. Story plot generation based on CBR. *Knowledge-Based Systems*, 18(4-5):235–242, August 2005.

10 Niels Kasch and Tim Oates. Mining script-like structures from the web. In *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading*, pages 34–42, Los Angeles, California, USA, 2010. Association for Computational Linguisitics.

11 Klaus Krippendorff. Testing the reliability of content analysis data: What is involved and why. In Klaus Krippendorff and Mary Angela Bock, editors, *The Content Analysis Reader*, pages 350–357. Sage Publications, Thousand Oaks, CA, USA, 2009.

12 J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, March 1977.

13 Edith Law and Louis von Ahn. *Human Computation*. Morgan & Claypool, 2011.

14 Boyang Li, Stephen Lee-Urban, George Johnston, and Mark O. Riedl. Story generation with crowdsourced plot graphs. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, Bellevue, Washington, USA, 2013. AAAI.

15 James Meehan. TALE-SPIN. In Roger C Schank and Christopher K. Riesbeck, editors, *Inside Computer Understanding*, pages 197–226. Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.

16 Matthew B. Miles and A. Michael Huberman. *Qualitative data analysis*. Sage Publications, Thousand Oaks, California, USA, 2nd edition, 1994.

17 Brian O'Neill. *A Computational Model of Suspense for the Augmentation of Intelligent Story Generation*. Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, 2013.

18 Brian O'Neill and Mark Riedl. Dramatis: A computational model of suspense. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, Quebec City, QC, Canada, 2014. AAAI Press.

19 Mark O. Riedl and R. Michael Young. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39(1):217–268, September 2010.

20 Johnny Saldaña. *The Coding Manual for Qualitative Researchers*. Sage Publications, Los Angeles, California, USA, 2009.

21 Roger C. Schank and R.P. Abelson. *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*, volume 2. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1977.

22 Avirup Sil and Alexander Yates. Extracting STRIPS representations of actions and events. In *Proceedings of the Recent Advances in Natural Language Processing*, pages 1–8, Hissar, Bulgaria, 2011.

23 Sigal Sina, Avi Rosenfeld, and Sarit Kraus. Generating content for scenario-based serious-games using CrowdSourcing. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, Quebec City, QC, Canada, 2014. AAAI Press.

24 Anselm L. Strauss. *Qualitative analysis for social scientists*. Cambridge University Press, Cambridge, UK, 1987.

25 Reid Swanson and Andrew Gordon. Say anything: A massively collaborative open domain story writing companion. In *Interactive Storytelling*, pages 32–40. Springer Verlag, 2008.

26 Robert Wilensky. PAM and micro PAM. In Roger Schank and Christopher Riesbeck, editors, *Inside Computer Understanding*. Erlbaum, Hillsdale, NJ, 1981.