StoryFramer: From Input Stories to Output Planning Models

StoryFramer: From Input Stories to Output Planning Models

Thomas Hayton[1], Julie Porteous[1], João F. Ferreira[1,3], Alan Lindsay[1], Jonathon Read[2]
[1]Digital Futures Institute, School of Computing, Teesside University, UK.
[2]Ocado Technology, Hatfield, UK.
[3]HASLab/INESC TEC, Universidade do Minho, 4704-553 Braga, Portugal.
firstinitial.lastname@tees.ac.uk | jonathon.read@ocado.com

Abstract

We are interested in the problem of creating narrative planning models for use in Interactive Multimedia Storytelling Systems. Modelling of planning domains has been identified as a major bottleneck in the wider field of planning technologies and this is particularly so for narrative applications where authors are likely to be non-technical. On the other hand there are many large corpora of stories and plot synopses, in natural language, which could be mined to extract content that could be used to build narrative domain models.

In this paper we describe an approach to learning narrative planning domain models from input natural language plot synopses. Our approach, called StoryFramer, takes natural language input and uses NLP techniques to construct structured representations from which we build up domain model content. The system also prompts the user for input to disambiguate content and select from candidate actions and predicates. We fully describe the approach and illustrate it with an end-to-end worked example. We evaluate the performance of StoryFramer with NL input for narrative domains which demonstrate the potential of the approach for learning complete domain models.

Introduction

Interactive Multimedia Storytelling (IS) systems allow users to interact and influence, in real-time, the evolution of a narrative as it is presented to them. This presentation can be via a range of different output media such as 2D or 3D animation (Mateas and Stern 2005; Porteous, Charles, and Cavazza 2013), filmic content (Piacenza et al. 2011) and text (Cardona-Rivera and Li 2016). In addition, a range of different interaction mechanisms have been used such as emotional speech input (Cavazza et al. 2009), gaze (Bee et al. 2010), and physiologicali measures (Gilroy et al. 2012).

AI planning has been widely used for narrative generation in IS as it provides: a natural "fit" with story plot lines represented as narrative plans; ensures causality which is important for the generation of meaningful and comprehensible narratives; and provides considerable flexibility and potential generative power. Consequently plan-based approaches have featured in many systems (e.g. as reported by (Aylett, Dias, and Paiva 2006; Riedl and Young 2010; Porteous, Charles, and Cavazza 2013)).

In this work we are interested in the problem of authoring the narrative planning domain models that are used in such IS systems. To date the authoring of narrative planning models has been handled manually, a common strategy being to build up the model via systematic consideration of alternatives around a baseline plot (Porteous, Cavazza, and Charles 2010b) and many prototype IS systems have sought inspiration from existing literary or filmic work. Examples include the Façade interactive drama which was based on "Who's Afraid of Virginia Woolf?" (Mateas and Stern 2005), The Merchant of Venice (Porteous, Cavazza, and Charles 2010a), Madame Bovary (Cavazza et al. 2009) and the tale of Aladdin (Riedl and Young 2010).

However this manual creation is extremely challenging. In this paper the problem we tackle is automation of narrative domain model creation. Our starting point is to look at natural language plot outlines as content from which to automatically induce planning models. We are developing a solution which takes natural language input (i.e. stories) and uses NLP techniques to construct structured representations of the text and keeps the user in the loop to guide refinement of the planning model. This approach represents an extension to the Framer system of (Lindsay et al. 2017) to application with narrative domain models. We have implemented the approach in a prototype system called StoryFramer.

In the paper we give an overview of the technical aspects of the approach and illustrate it with an end-to-end worked example using the tale of Aladdin taken from (Riedl and Young 2010). We present the results of an evaluation with a two domain models generated by StoryFramer and consider the potential of the approach.

Related Work

Some recent work in the area of automated domain model creation for planning has attempted to learn planner action models from natural language (NL) input.

Much of this work has attempted to map from NL input onto existing formal representations. For example, in relation to RoboCup@Home, Kollar et al. (2013) present a probabilistic approach to learning the referring expressions for robot primitives and physical locations in a region. Also Mokhtari, Lopes, and Pinho (2016) present an approach to

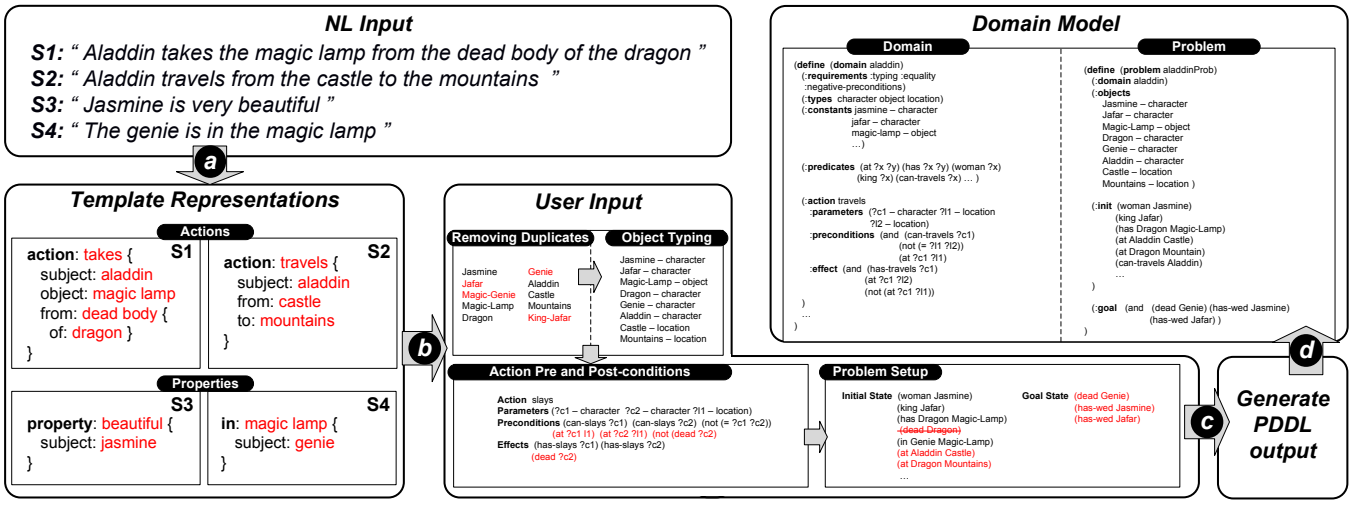

Figure 1: StoryFramer Overview: the NL input sentences and CoreNLP annotations are mapped to Template Representations ⓐ; the user disambiguates content, types objects and selects predicates for pre and post-conditions ⓑ; the different elements of the domain content are assembled ⓒ; and the domain content is output as PDDL domain model and problem file ⓓ.

learning action schemata for high-level robot control.

In (Goldwasser and Roth 2011) the authors present an alternative approach to learning the dynamics of the world where the NL input provides a direct lesson about part of the dynamics of the environment. Each lesson is supported by a small training data set to support learning from the lessons. In contrast to our approach, their system relies on a representation of the states and actions, which means their NLP approach can target an existing language.

More closely related to our work are attempts to learn planning models in the absence of a target representation. These include (Sil and Yates 2011) who used text mining via a search engine to identify documents that contain words that represent target verbs or events and then uses inductive learning techniques to identify appropriate action pre- and post-conditions. Their system was able to learn action representations, although with certain restrictions such as the number of predicate arguments. Branavan et al. (2012) introduce a reinforcement learning approach which uses surface linguistic cues to learn pre-condition relation pairs from text for use during planning. The success of the learnt model relies on use of feedback automatically obtained from plan execution attempts. Yordanova (2016) presents an approach which works with input text solution plans, as a proxy for instructions, and aims to learn pre- and post-condition action representations.

A similar increase in work aimed at automated creation has been seen in research in Narrative generation for Interactive Storytelling. However, an important difference with respect to narrative domains is that they do not share the same consistency and alignment with real-world domains as do more traditional benchmark planning domains. Hence approaches have tended to focus on (semi-)automated methods to gather story content, such as crowdsourcing, weblogs and story corpora. For example, crowdsourcing was used in: the SCHERAZADE system (Li et al. 2013) to acquire typical story elements that can be assembled as plot graphs and used in a process of story generation; SCENARIOGEN (Sina, Rosenfeld, and Kraus 2014) to gather a database of scenarios of everyday activities and likely replacements for use within a serious game context; and by (Nazar and Janssen 2010) for the hand annotation of logs from user sessions with the Restaurant Game for subsequent use in automating character interactions with human participants in a speech-based narrative setting. An alternative approach aims to obtain narrative content through mining of weblogs and story corpora. For example, SAYANYTHING (Swanson and Gordon 2012) selects narrative content on-the-fly from a corpora of weblogs in response to user text-based interaction, whilst (McIntyre and Lapata 2009) attempts to generate narratives using knowledge mined from story corpora for a particular genre.

Our work complements this, with input narrative content being mined from input natural language plot synopses.

StoryFramer Overview

Our approach to domain model generation is implemented in a system called StoryFramer, the main elements of which are shown in Figure 1. The system takes as input NL narrative synopses such as the outline story of Aladdin shown in Figure 2 and outputs a PDDL domain model and problem file such that the original story can be reproduced using a planner. The target output language is PDDL1.2 (Ghallab et al. 1998).

Currently the translation from NL to a domain model is a semi-automated process with the user in the loop for disambiguation of content at a number of stages. In this section we overview the main stages in this process and then illustrate it with an end-to-end example.

Extracting template representations from NL input

The first step in the approach is the generation of frame templates which are reduced representations of the input sen-

There is a woman named Jasmine. There is a king named Jafar. This is a story about how King Jafar becomes married to Jasmine. There is a magic genie. This is also a story about how the genie dies. There is a magic lamp. There is a dragon. The dragon has the magic lamp. The genie is confined within the magic lamp. There is a brave knight named Aladdin. Aladdin travels from the castle to the mountains. Aladdin slays the dragon. The dragon is dead. Aladdin takes the magic lamp from the dead body of the dragon. Aladdin travels from the mountains to the castle. Aladdin hands the magic lamp to King Jafar. The genie is in the magic lamp. King Jafar rubs the magic lamp and summons the genie out of it. The genie is not confined within the magic lamp. The genie casts a spell on Jasmine making her fall in love with King Jafar. Jasmine is madly in love with King Jafar. Aladdin slays the genie. King Jafar is not married. Jasmine is very beautiful. King Jafar sees Jasmine and instantly falls in love with her. King Jafar and Jasmine wed in an extravagant ceremony. The genie is dead. King Jafar and Jasmine are married. The end.

Figure 2: Aladdin outline plot from (Riedl and Young 2010)

tences. These templates capture the main action or property that a sentence is describing, as well as the objects mentioned and an indication of their roles within the sentence.

For this extraction we use Stanford CoreNLP (Manning et al. 2014), a publicly-available and widely-used annotation pipeline for natural language analysis. Of most relevance in this work are the syntactic parsing annotations that CoreNLP produces. Syntactic analysis in CoreNLP is a two-stage process. Firstly, phrase structure trees are generated using statistical analysis of datasets containing many examples of manually annotated sentence parses (Klein and Manning 2003). Secondly, these phrase structure trees are converted to dependency parse graphs using a series of manually-curated rules based on patterns observed in the phrase structure trees (de Marneffe, MacCartney, and Manning 2006). An example of the sort of dependency graphs that are output by CoreNLP is shown in Figure 3.

For our purposes the structure of these graphs must be further simplified to move closer to a predicate logic representation. This is achieved through a recursive set of rules that crawl the dependency graph, transforming the relations based on their types. CoreNLP use the Penn Treebank Project part-of-speech tags (Marcus, Marcinkiewicz, and Santorini 1993) to annotate the text. Most importantly, the root verb, subjects and objects form actions, predicates and domain objects as follows:

- The VBZ tag denotes verb (3rd person singular present) and this forms the basis of candidate action names. For example, the action `takes` in Figure 3
- The JJ tag denotes adjectives. These form the candidate properties e.g. the property `beautiful` in Figure 4.
- The NN tag (and variants NNS, NNP, NNPS) denote nouns; singular, plural, proper noun singular and proper noun plural respectively. These form the basis candidate objects (constants) for the domain. For example, `Aladdin` in Figure 3 and `Jasmine` in Figure 4.

Conjunctions in input sentences introduce new clauses, which themselves form further predicates. Other relation types such as modifiers and compounds are used to transform the names of the predicates and arguments.

Building Action Representations

Based on the CoreNLP annotations, StoryFramer creates a temporary action template which uses the verb as the action name and includes all the associated objects. An example template action for `takes` is shown in Figure 3. It can be observed that this template contains key elements of the action that will be output, namely, the name, arguments (the characters `aladdin` and `dragon`, and the `magic-lamp` object). We discuss this in more detail later (Worked Example section) following the rest of the StoryFramer overview.

Parameters For each action template the system labels each of the associated objects as candidate parameters for the output action. During the phase of user interaction these will be typed using the generic categories of: `character`, `object` and `location`.

Pre- and Post-conditions Following an approach similar to (Yordanova 2016), default predicates are added to the pre- and post-conditions of template actions, named `(can-`*action* `?x)` and `(has-`*action* `?x)` to introduce a baseline level of causality, sufficient to ensure generation of a baseline plan that corresponds to the original input NL story synopsis. For example, it may be necessary to use one of these predicates as part of the goal condition (this is the case with our Aladdin worked example).

Other predicates are added by the system as the PDDL domain files are output, following user interaction.

User Interaction

At this stage StoryFramer requires user interation and hence prompts for input to be used for the following:

- Removing Duplicates:

  Anytime the same object has been referred to in different ways in the NL input, the result is that the system finds multiple different objects. In this situation the user is asked to disambiguate. At the end of this stage every object should be represented by one unique identifier.
- Typing of Objects:

  The user is asked to sort the objects into types. So far in our experiments with narrative domains we have restricted this to the following small set of narrative categories: `character, object` and `location`.
- Action Pre and Post-conditions:

  The user is asked to select between possible pre- and post-conditions for inclusion in the domain model (i.e. "Do you want to include ...?"). These are of the following types:

NL *Aladdin takes the magic lamp from the dead body of the dragon.*

CoreNLP Annotation

```
[takes/VBZ
nsubj>Aladdin/NNP
dobj>[lamp/NN det>the/DT amod>magic/JJ]
nmod:from>[body/NN
  case>from/IN
  det>the/DT
  amod>dead/JJ
  nmod:of>[dragon/NN case>of/IN det>the/DT]]
punct./.]
```

Action Template

```
action :  takes {
subject :  aladdin
object :  magic lamp
from :  dead body
of :  dragon }
```

NL *Aladdin travels from the castle to the mountains.*

CoreNLP Annotation

```
[travels/VBZ
nsubj>Aladdin/NNP
nmod:from>[castle/NN case>from/IN det>the/DT
nmod:to>[mountains/NNS case>to/TO det>the/DT
punct./.]
```

Action Template

```
action :  travels {
subject :  aladdin
from :  castle
to :  mountains
```

Figure 3: Action Template Examples. The figure shows sample NL sentence input, with CoreNLP annotation and resulting action template after rewrite rules: from CoreNLP annotation, **verb**, **subject** and **object** of the sentence form the action name and arguments (see text for further details).

1) Predicates: the user is asked to select from identified predicates, such as `beautiful` in Figure 4, and use them to populate action preconditions where appropriate.

2) Locatedness: some conditions are commonly missing from the NL input relating to the location of characters and objects. For the work we present here we assume that all characters must always be "at" some location and that objects can be either "at" or in the possession of a character i.e. "has". Should such predicates be missing from the NL input, then users are asked to decide whether to include them.

3) Inequality: whenever an action template has multiple parameters of the same type the system assumes that these cannot be equal and prompts the user about inclusion of a `(not (= ?x1 ?x2))` precondition.

- Problem File setup:

  The final stage of user interaction is setting up a problem file. Every predicate detected by StoryFramer that was true at some time during the story represents a potential initial state fact or plan goal. The user is shown a list of facts and asked to delete those not appropriate, as well as adding any that were missed or not mentioned. For example, this frequently requires the selection of predicates relating to the location of characters in the initial state: something frequently missing from the input NL synopses.

  The user is also prompted to select suitable goal facts for the problem file. In our experiments we have used goal conditions which enable the generation of a plan which reproduces the story outline from the input NL story.

NL *Jasmine is very beautiful.*

CoreNLP Annotation

```
[beautiful/JJ
nsubj>Jasmine/NN
cop>is/VBZ advmod>very/RB
punct./.]
```

Property Template

```
property :  beautiful {
subject :  jasmine }
```

NL *The genie is in the magic lamp.*

CoreNLP Annotation

```
[lamp/NN
nsubj>[genie/NN det>The/DT]
cop>is/VBZ
case>in/IN
det>the/DT
amod>magic/JJ
punct./.]
```

Property Template

```
in :  magic lamp {
subject :  genie }
```

NL *The dragon is dead.*

CoreNLP Annotation

```
[dead/JJ
nsubj>[dragon/NN det>The/DT]
cop>is/VBZ
punct./.]
```

Property Template

```
property :  dead {
subject :  dragon }
```

Figure 4: Property Template Examples. The figure shows sample NL input sentences, the CoreNLP annotations for this sentence with the property and subject taken from the sentence adjective and noun (for further details see text).

PDDL output

The final stage in the process is the generation of the domain and problem file content which is output as PDDL.

Following the user interaction to provide type information for action parameters and domain objects the system may add additional parameters to actions at this stage. This is based on an assumption that all actions must have an associated location as this is needed in the longer term building up of a narrative domain: for example, for staging of generated story plans in a virtual environment. In practice we have observed that location details are frequently missing from synopses. Hence additional location parameters are automatically added to actions where there is no location parameter

associated with the action template from the input text.

For the domain file, the detail of the actions comes from the action templates with the types of action parameters added on the basis of the user input. Pre- and post-conditions are made up of predicates extracted from the input NL, and selected for inclusion by the user, along with system suggested predicates such as inequality testing.

Worked Example: Aladdin

In this section we present a worked example of the end-to-end process of planning domain model and problem file generation with StoryFramer. This example uses the NL plot synopsis from (Riedl and Young 2010) shown in Figure 2.

1) StoryFramer Processing

CoreNLP handles the input NL text, one sentence at a time and for each sentence returns the text with annotations. For example, for the following input sentences:

S1 "Aladdin takes the magic lamp from the dead body of the dragon"
S2 "Aladdin travels from the castle to the mountains"
S3 "Jasmine is very beautiful"
S4 "The genie is in the magic lamp"

the resulting CoreNLP annotations are as shown in Figures 3 and 4. For the action templates StoryFramer identifies the key action components and also adds the `can-X` and `has-X` predicates to action pre- and post-conditions to ensure a baseline for action causal chaining. Thus the outline actions from S1 and S2 at this stage are:

	Name	Precondition	Postcondition
S1:	`takes aladdin, magic-lamp, dragon`	`can-takes`	`has-takes`
S2:	`travels castle, mountain`	`can-travel`	`has-travel`

For the property templates, resulting from input sentence S3 and S4, the key components form the basis of predicates with name and arguments as follows:

	Name	Arguments
S3	`beautiful`	`jasmine`
S4	`in`	`magic-lamp, genie`

At the end of this first phase of automated processing with StoryFramer the sets of initial action templates, predicates and domain objects are as summarised in Figure 5.

2) User Input

Firstly, the user is asked to remove duplicate references to the same objects. For example, in Figure 5 the set of objects contains a number of duplicates, such as `Jafar` and `King-Jafar` and the user has selected unique object identifiers and the duplicates have been removed.

Once duplicates have been resolved the user sorts the set of objects into types. In our experiments to date we have restricted this to `character, object` and `location`. For the Aladdin story, the results of this phase of user interaction result in object typing as shown in Figure 5.

Actions	**Predicates**	**Objects**
`confined`	`at has woman`	`Jasmine`
`travels`	`king dead`	`Jafar`
`slays`	`knight in`	`Magic-Genie`
`takes gives`	`beautiful`	`Magic-Lamp`
`rubs casts`	`in-love`	`Dragon Genie`
`married sees`	`can-travels`	`Aladdin`
`wed`	`has-travels`	`Castle`
	`can-slays`	`Mountains`
	`has-slays`	`King-Jafar`
	`...`	

Resolved Objects

`(Jafar, King-Jafar)`	→ `Jafar`
`(Genie, Magic-Genie)`	→ `Genie`

Typing

`Jafar, Jasmine, Aladdin, ...`	→ `character`
`Castle, Mountain`	→ `location`
`Magic-Lamp`	→ `object`

Figure 5: Results of initial phase of automated StoryFramer processing: the figure shows the sets of names of actions, predicates and objects identified prior to user interaction. Duplicate Object References are highlighted (red) along with the results of user input to remove duplicates. Also shown are the results of user sorting of objects into types.

Next the user is prompted to select and reject predicates to populate the pre- and post-conditions of the output actions. These predicates are obtained as follows :

1. Predicates from properties in the input NL sentences. For Aladdin a selection of these are shown in Figure 5)
2. Locatedness predicates, `at` and `has`, introduced by StoryFramer if they are absent in the input NL and representing the location of objects of type `character` and `object`.
3. Predicates ensuring unique object grounding of multiple parameters of the same type, i.e. inequality.

The action `takes` in Figure 6 shows examples of the results of user selection of these predicates and the building of the output action.

The final phase of user interaction is selection of predicates for the setting up of a problem file: the initial state and the goal conditions. In our experiments the user selected those predicates from the initial state and goal conditions which allowed us to regenerate the plan corresponding to the input NL plot synopsis.

3) PDDL Output

The final step of the process is the outputting of the Domain Model and Planning Problem as PDDL files. For this example these can be found online:

`https://drive.google.com/drive/folders/0B6Rv1Q3KYqtMcXhUMFFSZzh1a1U?usp=sharing`

```
(:action takes
   :parameters
      (?c1 ?c2 - character ?o - object ?l - location)
   :precondition (and
1     (dead ?c2)
2     (at ?c1 ?l1) (at ?c2 ?l1) (has ?c2 ?o1)
      (can-takes ?c1) (can-takes ?c2) (can-takes ?o1)
3     (not (= ?c1 ?c2)))
   :effect (and
1     (has ?c1 ?o1) (not (has ?c2 ?o1))
      (has-takes ?c1) (has-takes ?c2) (has-takes ?o1) )))
```

Figure 6: Example of StoryFramer Building for action `takes`. Following user input the parameter object names from the input NL have been replaced by variables of the appropriate types. Precondition predicates have been: selected by the user 1; system suggested locatedness and chaining predicates have been retained by the user 2; the system has introduced inequality tests for objects of the same type 3. For the postconditions the user has retained locatedness and chaining predicates as shown 2.

Evaluation

In this section we present an evaluation to assess how accurate StoryFramer is. We evaluate StoryFramer with two domains: the tale of Aladdin taken from (Riedl and Young 2010) and an old American West story taken from (Ware 2014). We selected these two sources because they provide natural language descriptions that we can use as input and they include planning domains that we can use to compare with the domains generated by StoryFramer. In particular, we evaluate StoryFramer by comparing the set of recognised actions and predicates with the actions and predicates used in the selected domains.

The domains used in the evaluation can be found online using the link provided at the end of the worked example.

Domain 1: The tale of Aladdin

We used StoryFramer with two texts describing the tale of Aladdin. Both texts were taken from (Riedl and Young 2010): one is shown here, in Figure 2; the other is a variation (see Figure 13 of Riedl's paper). Table 1 shows that all actions but one were recognised by StoryFramer. When compared to Riedl and Young's planning domain, only one action is not recognised. All the other 11 actions are recognised with four of them being named based on the same verb. The action `marry` is recognised twice: as `wed` and as `married`.

In terms of predicates, out of Riedl and Young's 24 predicates, only two predicates that are mentioned in the text are not recognised by StoryFramer: the binary predicates `married-to` and `loyal-to` (however, a unary predicate `loyal` is recognised). Nine (9) are recognised in exactly the same way: one (1) as a type, one (1) as an object, one (1) as a constant, and six (6) as predicates. Eight (8) are recognised, but with a different name from the one that Riedl and Young used: four (4) of them are minor variations (e.g. `married`/`has-married` and `loves`/`in-love`); two (2) of them are recognised as words that appear in the text: instead of `alive` and `female`, the system recognised `dead` and `woman` (resp.); and two (2) of them are recognised as types (Riedl and Young used `thing` and `place` instead of `object` and `location`; in both cases, none of the words appear in the text). Finally, there are seven (7) predicates that were not recognised: five (5) of them do not appear in the text (`scary`, `monster`, `male`, `single`, and `intends`); and finally, as mentioned above, two (2) are mentioned in the text, but are not recognised.

Output Plan	Corresponding NL sentences
(sees jafar jasmine castle)	King Jafar sees Jasmine and instantly falls in love with her.
(travels aladdin castle mountains)	Aladdin travels from the castle to the mountains.
(slays aladdin dragon mountains)	Aladdin slays the dragon.
(takes aladdin dragon magic-lamp mountains)	Aladdin takes the magic lamp from the dead body of the dragon
(travels aladdin mountains castle)	Aladdin travels from the mountains to the castle.
(gives aladdin jafar magic-lamp castle)	Aladdin hands the magic lamp to King Jafar.
(rubs jafar genie magic-lamp castle)	King Jafar rubs the magic lamp and summons the genie out of it.
(casts genie jasmine jafar castle)	The genie casts a spell on Jasmine making her fall in love with King Jafar.
(slays aladdin genie castle)	Aladdin slays the genie.
(wed jafar jasmine castle)	King Jafar and Jasmine wed in an extravagant ceremony.

Figure 7: Output Plan and Corresponding Input NL sentences for the tale of Aladdin: on the left hand side are the 10 actions in the output plan generated using the learned domain model; alongside each action (right hand side) are the corresponding input sentences from the original story.

Output Plans We used the StoryFramer generated domain and problem file from the original Aladdin NL input to generate an output narrative plan (using METRIC-FF (Hoffmann and Nebel 2001)). The plan consists of the 10 actions which are shown in Figure 7, along with corresponding input.

Domain 2: An old American West story

We also used StoryFramer with natural language sentences taken from (Ware 2014). These are are part of an old American West story about how a young boy named Timmy is saved (or not saved) from a deadly snakebite. His father, Hank, can save him by stealing antivenom from Carl, the town shopkeeper, but this theft causes sheriff William to hunt down Hank and dispense frontier justice.

In the thesis Ware gives seven example solution plans and translations of them into NL. It is these NL sentences which we used as input to StoryFramer. We list them here and show

Riedl and Young (2010)	Recognised by StoryFramer
travel	✓(travels)
slay	✓(slays)
pillage	✓(takes)
give	✓(gives)
summon	✓(rubs)
love-spell	✓(casts)
marry	✓(wed,married)
fall-in-love	✓(sees)
order	✗
command	✓(uses)
appear-threatening	✓(appears)

Table 1: When compared to Riedl and Young's planning domain, only one action is not recognised by StoryFramer. All the other 11 actions are recognised with four of them being named based on the same verb. The action 'marry' is recognised twice: as 'wed' and as 'married'.

in brackets the action names used by Ware:

- *Timmy died. (die)*
- *Carl the shopkeeper healed Timmy using his medicine. (heal)*
- *Hank shot his son Timmy. (shot)*
- *Hank stole antivenom from the shop, which angered Sheriff William. (steal)*
- *Hank healed his son Timmy using the stolen antivenom. (heal)*
- *Sheriff William shot Hank for his crime. (shoot)*
- *Hank intended to heal his son Timmy using the stolen antivenom. (heal)*
- *Sheriff William intended to shoot Hank for his crime. (shoot)*
- *Hank got bitten by a snake. (snakebite)*
- *Hank intended to heal himself using the stolen antivenom. (heal)*

In Table 2, we show that all the actions used by Ware were recognised by StoryFramer. Note that whilst Ware used the action `heal` to model the actions mentioned in the sentences "*Hank healed...*" and "*Hank intended to heal*", StoryFramer recognised two different actions: `heal` for the first sentence and `intended` for the second.

In terms of predicates, results were not so good as with the tale of Aladdin. The predicates generated by StoryFramer are based on the text provided as input, so, besides predicates common in general narratives (e.g. `at ?c ?l` or `has ?c ?o`), StoryFramer generated predicates associated with the recognised actions (e.g. `has-died`, `can-shot`, and `has-bitten`). It also introduced as constants all the characters mentioned (`Hank`, `Timmy`, `Carl`, and `Sheriff`) and some objects and locations used in the narrative (`Medicine`, `Antivenom`, and `Shop`). On the other hand, Ware introduced a predicate `status ?p ?s` that is to be used with one of three constants: `Healthy`, `Sick`, or `Dead`. He also introduced predicates `owns ?c ?o`, `armed ?c`, and `parent ?c1 ?c2`.

Ware (2014)	Recognised by StoryFramer
die	✓(died)
heal	✓(healed)
shoot	✓(shot)
steal	✓(stole)
snakebite	✓(bitten)
✗	intended

Table 2: All the actions used by Ware (2014) were recognised. StoryFramer also recognised an additional action ('intended').

This mismatch is justified because Ware is using predicates that are not mentioned explicitly in the sentences that we used as input. For example, the sentences do not make any reference to the words or states Healthy and Sick. We discuss how this limitation can be addressed in the section on future work.

Output Plans Even though there is a clear mismatch between the predicates recognised by StoryFramer and the ones used by Ware, the generated domains can be used to produce all the seven plans listed for the Western Domain (see (Ware 2014, Fig. 3.3) and Figure 8). However, we note that for two of these plans, F and G, it was necessary to remove reasoning about intent: despite StoryFramer correctly generating an `intends` action from the NL input. This is because intention reasoning (a feature of Ware's COPCL planner and other narrative planners in the tradition of the IPOCL planner of (Riedl and Young 2010)) requires the use of a planner capable of intentional reasoning which is beyond the scope of our current work.

With intention removed we were able to generate the same plans as reported by Ware. The results are shown in Figure 8. Although we note that in order to reproduce the same ordering of actions it was necessary to use intermediate goals as described in (Porteous, Cavazza, and Charles 2010b). For example, to enforce the ordering that a goal `(has-shot sheriff hank)` occurs before another goal `(has-died timmy)` the problem is written in PDDL3.0 using the modal operator `sometime-before`[1] and the plan is generated using a decomposition approach that solves each subgoal in turn.

Conclusions and Future Work

In the paper we have presented an overview of our approach to automated domain model generation and its implementation in the prototype system StoryFramer. We assessed the performance of the approach on a couple of publicly available narrative planning domains, for which narrative synopses are also available.

An important aspect of the approach is that it is possible to go from NL input to an output domain model and problem instance, with which it is possible to generate a plan that

[1]The semantics of `(sometime-before A B)` requires that application of actions in solution plans make `B` true before `A`.

Plan	**Ware Plans**	**StoryFramer Plans**
A	(die Timmy)	(died timmy shop)
B	(heal Carl Timmy)	(healed carl-shopkeeper timmy medicine shop)
C	(shoot Hank Timmy)	(shot timmy hank shop)
D	(steal Hank Antivenom Carl William) (heal Hank Antinvenom Timmy)	(stole hank sheriff-william antivenom shop) (healed hank timmy antivenom shop)
E	(steal Hank Antivenom Carl William) (heal Hank Antinvenom Timmy) (shoot William Hank)	(stole hank sheriff-william antivenom shop) (healed hank timmy antivenom shop) (shot sheriff-william hank shop)
F	(steal Hank Antivenom Carl William) (shoot William Hank) <heal Hank Antivenom Timmy> (die Timmy)	(stole hank sheriff-william antivenom shop) (shot sheriff-william hank shop) - (died timmy shop)
G	(steal Hank Antivenom Carl William) <shoot William Hank> (snakebite Hank) <heal Hank Antivenom Hank> (heal Hank Antivenom Timmy)	(stole hank sheriff-william antivenom shop) - (bitten hank shop) - (healed hank timmy antivenom shop)

Figure 8: Comparison of output plans for the Western Domain from (Ware 2014): those listed by Ware and corresponding to the NL input to StoryFramer; and those generated by StoryFramer ignoring intent. See text for detail.

corresponds to the original NL input. Much of this process is automated but user input may be required for some aspects such as disambiguation of content. However there is scope to further automate the process as part of the future work.

Amongst our plans for future work we are keen to exploit further the part-of-speech information provided by CoreNLP in combination with other linguistic resources in order to disambiguate content. We also intend to explore the use of a commonsense reasoning engine which would enable inference of aspects such as family and social relationships (e.g. from references such as "parent" in the input).

There are also possibilities to combine this with reasoning that is able to automatically extend an existing domain model, for example via the use of antonyms to find opposite actions, as in (Porteous et al. 2015).

We may also look to use multiple story synopses as input to incrementally build up a large domain ontology. This could for example be used to learn actions from multiple episodes of a series so that generated output plans can show more variation.

References

Aylett, R.; Dias, J.; and Paiva, A. 2006. An Affectively Driven Planner for Synthetic Characters. In *Proc. of 16th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Bee, N.; Wagner, J.; André, E.; Charles, F.; Pizzi, D.; and Cavazza, M. 2010. Multimodal interaction with a virtual character in interactive storytelling. In *the 9th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2010*, 1535–1536.

Branavan, S. R. K.; Kushman, N.; Lei, T.; and Barzilay, R. 2012. Learning High-level Planning from Text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, ACL '12, 126–135. Stroudsburg, PA, USA: Association for Computational Linguistics.

Cardona-Rivera, R. E., and Li, B. 2016. PLOTSHOT: Generating Discourse-constrained Stories Around Photos. In *Proceedings of the 12th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Cavazza, M.; Pizzi, D.; Charles, F.; Vogt, T.; and André, E. 2009. Emotional input for character-based interactive storytelling. In *Proc. of the 8th Int. Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

de Marneffe, M.-C.; MacCartney, B.; and Manning, C. D. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Conference on Language Resources and Evaluation*, 449–454.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; ; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. http://www.inf.ed.ac.uk/teaching/courses/propm/papers/pddl.html.

Gilroy, S. W.; Porteous, J.; Charles, F.; and Cavazza, M. 2012. Exploring Passive User Interaction for Adaptive Narratives. In *Proc. of the 17th Int. Conf. on Intelligent User Interfaces (IUI-12)*.

Goldwasser, D., and Roth, D. 2011. Learning from natural instructions. In *Proc. of the 22nd Int. Joint Conf. on Artifical Intelligence (IJCAI)*.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation throughHeuristic Search. *Journal of AI Research* 14:253–302.

Klein, D., and Manning, C. D. 2003. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. 3–10.

Kollar, T.; Perera, V.; Nardi, D.; ; and Veloso, M. 2013. Learning Environmental Knowledge from Task-based

Human-robot Dialog. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Li, B.; Lee-Urban, S.; Johnston, G.; and Riedl, M. 2013. Story Generation with Crowdsourced Plot Graphs. In *Proc. of the 27th AAAI Conf. on Artificial Intelligence*.

Lindsay, A.; Read, J.; F. Ferreira, J.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning Models from Natural Language Action Descriptions. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS)*.

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J. R.; Bethard, S.; and McClosky, D. 2014. The stanford corenlp natural language processing toolkit. In *The Annual Meeting of the Association for Computational Linguistics (System Demonstrations)*, 55–60.

Marcus, M. P.; Marcinkiewicz, M. A.; and Santorini, B. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Mateas, M., and Stern, A. 2005. Structuring Content in the Façade Interactive Drama Architecture. In *Proc. of the 1st Conf. on AI and Interactive Digital Entertainment (AIIDE)*.

McIntyre, N., and Lapata, M. 2009. Learning to Tell Tales: A Data-driven Approach to Story Generation. In *Proceedings of 47th Meeting of the Association for Computational Linguistics (ACL)*.

Mokhtari, V.; Lopes, L. S.; and Pinho, A. J. 2016. Experience-Based Robot Task Learning and Planning with Goal Inference. In *Proc. of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*.

Nazar, R., and Janssen, M. 2010. Combining resources: Taxonomy extraction from multiple dictionaries. In *Proc. of the 7th Int. Conf. on Language Resources and Evaluation*.

Piacenza, A.; Guerrini, F.; Adami, N.; Leonardi, R.; Teutenberg, J.; Porteous, J.; and Cavazza, M. 2011. Changing video arrangement for constructing alternative stories. In *Proc. of the 19th ACM International Conference on Multimedia*.

Porteous, J.; Lindsay, A.; Read, J.; Truran, M.; and Cavazza, M. 2015. Automated Extension of Narrative Planning Domains with Antonymic Operators. In *Proc. of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Porteous, J.; Cavazza, M.; and Charles, F. 2010a. Narrative Generation through Characters' Point of View. In *Proc. of 9th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS 2010)*.

Porteous, J.; Cavazza, M.; and Charles, F. 2010b. Applying Planning to Interactive Storytelling: Narrative Control using State Constraints. *ACM Transactions on Intelligent Systems and Technology (ACM TIST)* 1(2):1–21.

Porteous, J.; Charles, F.; and Cavazza, M. 2013. NetworkING: using Character Relationships for Interactive Narrative Generation. In *Proc. of 12th Int. Conf. on Autonomous agents and multi-agent systems (AAMAS)*. IFAAMAS.

Riedl, M. O., and Young, R. M. 2010. Narrative Planning: Balancing Plot and Character. *Journal of AI Research* 39:217–267.

Sil, A., and Yates, A. 2011. Extracting strips representations of actions and events. In *Recent Advances in Natural Language Processing (RANLP)*.

Sina, S.; Rosenfeld, A.; and Kraus, S. 2014. Generating content for scenario-based serious games using CrowdSourcing. In *Proceedings of 28th AAAI Conference on Artificial Intelligence (AAAI)*.

Swanson, R., and Gordon, A. S. 2012. Say Anything: Using Textual Case-Based Reasoning to Enable Open-Domain Interactive Storytelling. *ACM Trans. Interact. Intell. Syst.* 2(3).

Ware, S. G. 2014. *A plan-based model of conflict for narrative reasoning and generation*. Ph.D. Dissertation.

Yordanova, K. 2016. From Textual Instructions to Sensor-based Recognition of User Behaviour. In *Proc. of 21st Int. Conf. on Intelligent User Interfaces*, IUI Companion. ACM.