Chapter 4

Scheherazade

A set of proposed discourse relations such as the SIG is more useful when implemented as a machine-readable markup scheme and applied to a corpus with automatic or manual annotation. In Chapter 3, we developed a novel set of relations for representing the temporal, causal, affectual and goal-oriented features of a narrative discourse. In this chapter, we describe the implementation of a software package that facilitates story annotation, representation and management, using the SIG formalism as the basis of its data structure.

The system we have built, SCHEHERAZADE, meets six design goals:

1. Well-formed SIG encodings: SCHEHERAZADE allows a user to interactively build and construct encodings so that the semantics are enforced (for example, that provides and damages arcs can only point to Affect nodes).

2. Computability: The system is able to perform inference on SIG encodings according to the logical entailment rules that we outline in Chapter 3 and Appendix C, such as tracing whether an interpretative state is Actualized, Prevented/Ceased or Hypothetical, and inferring indirect causes based on causal chains.

3. Scalable precision: SCHEHERAZADE allows annotators to build propositional equivalents of story clauses and sentences. Specifically, it provides a process for encoding predicate-argument structures that leverage the taxonomies of nouns and verb frames found in external linguistic resources.

4. Domain independence: We strove to avoid over-committing the knowledge base (or the discourse model) to a particular set of narratives or a particular narrative genre.

5. Extensibility: We built the system as a platform for story management, so that other discourse relations could be applied to the same text by means of an API. The API also allows external learning tools to extract features from encodings.

6. Accessibility: We built an interactive, graphical user interface so that trained annotators can construct encodings from source texts. This includes a feedback text generator that serializes the encodings back into surface text, for purposes of allowing annotators to check whether the system has correctly captured the intended meaning of the story.
The following sections provide details regarding the design and implementation of the system. We describe the system architecture in Section 4.1 and the core logic in Section 4.2. Section 4.3 delves into the graphical interface we have developed for community story annotation. We then describe in Section 4.4 the textual generation module which “reverses” the annotation process by synthesizing a discourse from an encoding. We conclude in Section 4.5, leaving the experimental collection project to Chapter 5.

4.1 Data Structure and Architecture

The data structure for a narrative in SCHEHERAZADE mirrors the formal description of a SIG we gave in Chapter 3. Specifically, we use a graph structure where nodes represent elements such as spans of surface discourse, states in time, goals of characters, actions that occur in the timeline, and so on. The content of P and I nodes can either be placeholder content, where only the agent is indicated, or a more complete propositional modeling consisting of predicate-argument structures tied to a series of external resources. For instance, instead of placing “John walked to the store” in a node, we place \texttt{walk(person1, store1)} where \texttt{walk(<agent>, <destination>)} is a verb frame from a formal taxonomy of such frames, and \texttt{person1} and \texttt{store1} invoke instances of a \texttt{man} and \texttt{store} respectively (which are noun types from another formal taxonomy).

The “optional” nature of propositional modeling allows us to encode precise information about events and statives when possible, but still have a well-formed SIG encoding otherwise. On one hand, state-of-the-art semantic parsing tools are currently unable to automatically convert a text into a sequence of predicate-argument structures with high accuracy, and in our formative evaluations, even trained human users can find the same task challenging. On the other hand, when propositional modeling is present in an encoding, it is a rich source of structured data with which we might algorithmically find similarities and analogies between stories. SCHEHERAZADE supports either approach, and in Chapter 5 we explore the nature of the trade-off in detail by collecting corpora of story encodings under both conditions and comparing the results.

The system distinguishes between three classes of data (Figure 4.1):

**Narrative semantics.** This class includes the definitions and logical rules of the SIG schemata as we defined it in Chapter 3. It is a “hard” constraint in the data structure, in that it is immutable over all stories and all domains. SCHEHERAZADE enforces the rules of the SIG and returns an error to the user if an illegal change is requested during the annotation process. For example, there must always be a Reality timeline and zero or more alternate timelines, and circular plans are illegal.

**World knowledge** refers to the particular verb frames (predicates), noun types, and other facets of linguistic knowledge which are available for propositional modeling during the annotation process. It also includes a list of legal Affect types, such as \texttt{FREEDOM} and \texttt{EGO}, as we discussed in Section 3.3.2.8. For instance, there might be a \texttt{say} predicate encoded in this layer that, like a frame [\textit{Minsky, 1975}], is known to take two type-restricted arguments with particular thematic roles (a conscious entity as a speaker in the Agent role, and an object as an intended hearer in the Experiencer role). This
Figure 4.1: Three classes of data are distinguished by Scheherazade, each of which applies the one that appears beneath.

is a “soft” constraint, in that the system can be configured prior to the annotation process with a supply of predicate frames, noun types, selectional restrictions and Affect types that are to exist in the story-world. In the case of say, it would then disallow a non-organism from serving as an Agent. Similarly, a configuration of world knowledge geared toward Aesop’s fables need not include out-of-domain knowledge such as American. We will soon describe a default knowledge base that we have compiled for our experiments involving some 200,000 noun and verb elements.

**Story content** includes the content of a particular encoding of a source text. Propositions, alternate timelines, goals, plans, beliefs and Affect nodes, once instantiated in an iterative annotation process, are linked back to nodes that represent spans of source text (the textual layer). The content must follow the constraints of both narrative semantics and world knowledge.

The result is a data structure that includes the textual layer, timeline layer and interpretative layer of a complete SIG encoding. Figure 4.2 illustrates the way that the three classes of knowledge interact. Narrative semantics, world knowledge and story content are visualized as black, grey, and white shapes, respectively. Nouns and predicate frames are first stored as world knowledge (in grey), then instantiated (in white) and linked to particular clauses in the source text.

As an implementation of a descriptive model, rather than a prescriptive model, the system does not try to understand whether the content of any action or stative makes logical “sense” in the context of the story. For example, it will allow a character to act even if the “die” predicate is applied to the same character at a previous story state. However, as we explore in Chapter 5, it will leverage the structure found in the external taxonomies to find semantic similarities between story propositions.
System Architecture

The process of symbolically encoding a story in SCHEHERAZADE begins with the loading of a text file containing the source text. A user then repeatedly sends various commands to a command interpreter to build up the data structure—instructions to establish a new alternate timeline, add a new State node, instantiate a new proposition with a certain predicate frame and set of arguments, link a Proposition node to a plan with an actualizes arc, and so on. The user receives acknowledgment that each command has been carried out, and that the encoding remains valid on both the structural and content levels.

An architectural overview of SCHEHERAZADE is shown in Figure 4.3. At the bottom of the stack is a general-purpose engine for managing semantic networks. This engine can be configured to accept arbitrary types of nodes, arcs and attachment/inference rules. It also includes a serializer for saving networks to disk and a parser for reading them back into memory. A separate module, the Story Logic Manager (SLM), applies the particular logical form of the SIG network structure, including node types, arc types and the rules governing their inter-relationships. The SLM also incorporates the default knowledge base we have compiled from external linguistic resources.

The Story Logic Manager is exposed to plug-ins and third-party tools by means of an API. Our intention is for future work to use SCHEHERAZADE as a foundational platform for work in narrative analysis. In Chapter 5, we use the API to extract features of SIG encodings for purposes of finding similarities and analogies between story encodings.

Finally, we have built a graphical annotation interface on top of the API, iterating...
the design over the course of several formative evaluations. This involves a separate text generation module. As we have mentioned, this module “reverses” the encoding process, serializing a story encoding into text in order to provide annotators with helpful feedback.

The following sections go into more detail about each of these components.

4.2 Semantic Network Engine and Story Logic Manager

At the core of SCHEHERAZADE is a general-purpose knowledge representation engine that allows for the iterative construction of semantic networks, including constraint satisfaction and first-order logic entailments. Each node contains a frame type or frame instance, and each arc represents a first-order relation. The engine functions both as a tool for expressing logical relations and as a robust database for storing world knowledge and story content on the order of several hundred thousand connected nodes [Elson and McKeown, 2009].

We developed this engine to be separate from the Story Logic Manager; it has customizable data types and inference rules so that one can use it for purposes other than story logic. For instance, an API allows users to customize rules for entailment, deletion and typing, which it will then enforce as the network is built, modified, saved to disk and loaded from disk (Table 4.1). We have released the engine, along with the rest of the SCHEHERAZADE library, as a public resource.¹

The Story Logic Manager (SLM) rests above the semantic network engine and imbues it with the particular logical constraints of the SIG: the types of nodes found in the three layers,

¹http://www.cs.columbia.edu/~delson
Parameterized Rule Name | Meaning
--- | ---
circularLinksAllowed | \( f(a, b) \land f(b, c) \land f(c, a) \) is legal
toNodesInheritFromSubtypes | \( f(a, b) \land isa(a, c) \land f(c, b) \)
toNodesInheritFromSupertypes | \( f(a, b) \land isa(c, a) \land f(c, b) \)
toNodesInheritFromSubtypes | \( f(a, b) \land isa(b, c) \land f(a, c) \)
toNodesInheritFromSupertypes | \( f(a, b) \land isa(c, b) \land f(a, c) \)
multipleForwardLinks | \( f(a, b) \land f(c, b) \) is legal
multipleBackwardLinks | \( f(a, b) \land f(c, b) \) is legal
validLeftType | \( f(a, b) \land isType(a, t) \) is legal
validRightType | \( f(a, b) \land isType(b, t) \) is legal
mustHaveSameType | \( f(a, b) \land isType(a, t) \land \neg isType(b, t) \) is legal

table 4.1: Rules for entailment, deletion, and typing parameterized by the semantic network engine for each arc type (function \( f(a, b) \)).

<table>
<thead>
<tr>
<th>Construction</th>
<th>Destruction</th>
<th>Retrieval &amp; Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignEvent</td>
<td>modifyEventTime</td>
<td>findStoryIntersections</td>
</tr>
<tr>
<td>assignModifier</td>
<td>modifyAssociatedText</td>
<td>getEventsBeginningAt</td>
</tr>
<tr>
<td>assignStatic</td>
<td>modifyEvent</td>
<td>getEventsEndingAt</td>
</tr>
<tr>
<td>assignInterpretativeNode</td>
<td>modifyModifier</td>
<td>getEvents</td>
</tr>
<tr>
<td>defineEventFrame</td>
<td>modifyStatic</td>
<td>getStatics</td>
</tr>
<tr>
<td>defineModifierFrame</td>
<td>removeEvent</td>
<td>getInterpretativeCausalChains</td>
</tr>
<tr>
<td>defineStaticFrame</td>
<td>removeModifier</td>
<td>getInterpretativeNodes</td>
</tr>
<tr>
<td>linkInterpretativeNodes</td>
<td>removeStaticarc</td>
<td>getLinkedInterpElements</td>
</tr>
<tr>
<td>defineNoun</td>
<td>redo/undo</td>
<td>getDefinedNouns</td>
</tr>
<tr>
<td>newAlternateTimeline</td>
<td>revert</td>
<td>queryForPattern</td>
</tr>
</tbody>
</table>

Figure 4.4: A subset of the commands offered by the Story Logic Manager’s API.

how timelines must be structured, the rules for determining whether an interpretative-layer node is actualized at some point in the Reality timeline, and so on. The SLM provides an API for higher-level tools, using SCHEHERAZADE as a software library, to construct, store and load encodings. A subset of the commands offered by the API is shown in Table 4.4. For each command, the SLM raises an error if the request is invalid (e.g., a user tries to delete a non-existent node); there are particular errors if the request violates the SIG schemata (such as when one attempts to create a cyclical plan). It then interprets each command into a sequence of operations for the semantic network engine to carry out.

The commands listed in Table 4.4 involve the definition and assignment of four types of world knowledge. “Definition” commands augment the world knowledge structure with a new frame or noun type, while “assignment” commands instantiate a frame or type into an instance node and add the result to the network as story content. For instance, we may first define person as an organism and store as a location, then walk(<agent>,


As a verb frame. We may then assign John as a particular person, countryStore as a particular store, and walk(john, countryStore) as a timeline event in which John walks to a country store. The SLM enforces these selectional restrictions, finding that an organism is a satisfactory agent and a location is a satisfactory destination.

In general, the world knowledge structure involves four taxonomies, one for each facet of linguistic knowledge available for composing propositions:

1. **Nouns.** We identify five classes of nouns:
   - *character*, an animate being capable of agency
   - *location*, a relative or absolute spatial placement
   - *prop*, a non-agentive physical object
   - *activity*, a behavior such as a gathering or performance
   - *quality*, an attribute such as “handsomeness” or “height”

   There is a hierarchical taxonomy for each class. For instance: *monkey IS-A primate*, a type of character.

2. **Statives.** Statives are predicate frames that represent *durative* properties of nouns, which apply over a span of time but do not transform the story-world from one state to another. Adjectival descriptions (such as happy(<character>)) are statives, as are abilities, amounts, comparisons, beliefs, fears, goals, plans, hopes, identities (“John was the masked assailant”), obligations, possessions, and positional relationships (“the book was on the table”).

3. **Events.** Events are predicate frames that represent state-changing actions and happenings. The frame slots can be filled with nouns, as in walk taking an agent and a destination, or with nested propositions. An example of the latter would be says(<character>, <stative>), which represents the statement of some stative by a character (“John said that he was hungry”).

4. **Modifiers.** Modifier frames reference other propositions. slowly(<action>), for example, fills its slot with a reference to an action that is happening slowly.

For purposes of our experiments in Chapter 5, we turned to external linguistic resources to populate these taxonomies. Nouns, statives and modifiers adapt WordNet [Fellbaum, 1998], a well-established lexicon that features thousands of words organized into *synsets* with the same meaning. One synset, for example, includes the nouns “meadow” and “hayfield” in their typical senses. Synsets are organized into hypernym trees, with each synset related to more and less specific synsets. As our knowledge model also involves hierarchical taxonomies of nouns, we needed only to decide which subtrees of the root noun synset (entity) were to be adapted for each Scheherazade noun type. For example, to populate our list of available character types we adapted the *organism* subtree, allowing users to model stories concerning thousands of animal species or roles such as traveler. Table 4.2 lists the roots of the subtrees adapted for each noun class. Overall, we adapted 29 WordNet subtrees,
including approximately 125,000 total noun lexemes (including multiple instances of nouns that appear in more than one adapted synset).

WordNet’s adjectives, meanwhile, serve as adjectival statives that describe people or things (“the king was mighty”); its adverbs provide the basis for modifiers (“the king apologized graciously”). Other modifiers were hand-authored to serve as connectives between the modified proposition and a “third party” proposition: “A because/ despite/ in order to B.” We implemented a routine that allows the user to either model a new proposition for the connected B clause, or invoke (plug in) a proposition that occurred elsewhere on the Reality timeline. In other words, a “nested proposition” that serves as an argument can be a reference to an actual event that occurred elsewhere in the story—“The king died. The queen died because the king had died.”

While WordNet provides a hyponym tree for verbs as well, there is limited information about the manner in which each verb can be used as a predicate. For thematic roles, selectional restrictions, and syntactic descriptions, we turned to VerbNet [Kipper et al., 2006], the largest online verb lexicon currently available for English. Each verb is annotated with thematic roles (arguments) and their selectional restrictions, as well as syntactic frames for the various sentence constructions in which the verb and its roles might appear. Figure 4.5 illustrates through example the adaption of a VerbNet record into Scheherazade world knowledge. For each lexeme that serves as a member of the VerbNet class, we:

1. Find the corresponding WordNet synset for the member, which allows us to arrange predicates into an IS-A hierarchy (e.g., bash IS-A hit);
2. Map each VerbNet thematic role into a SCHEHERAZADE thematic role (such as a restriction to a Patient becoming a restriction to a prop noun);

3. Generate an easily readable phrase to represent the predicate in the user interface (such as “something bashes something with something”); and

4. Map each VerbNet syntactic frame into a distinct predicate frame. A syntactic frame may include only some, not all, of the thematic roles defined by the WordNet record. The syntactic construction that VerbNet gives is adapted into a plan for our textual generation module (Section 4.4).

We implemented SCHEHERAZADE so that a frame’s thematic role can carry a “slot restriction” of either a noun, a nested proposition (an inner event, stative or modifier) or a reference to an alternate timeline. The exact mappings with which we populated slot restrictions with the information from VerbNet records are given in Tables 4.3 and 4.4. For each table, a set of VerbNet thematic roles and selectional restrictions are given in the first column, and a set of syntactic restrictions are given in the second column. If a VerbNet slot featured at least one of the thematic roles or one of the selectional restrictions, and satisfied the syntactic restrictions (if any), it was mapped into the SCHEHERAZADE slot with the restriction given in the third column. Table 4.3 gives the mappings for noun restrictions,
while Table 4.4 gives those for nested-proposition and timeline restrictions. In the latter case, slot restrictions carry a grammatical component: A SCHEHERAZADE frame may call for a nested stative in the assertive mode (“she told him that he was gracious”) or in the imperative mode (“she told him to be gracious”), among others.

In the cases of control verbs, an argument argument of the nested proposition (that is, the subordinate clause) is controlled by the encapsulating frame (the main clause). The system recognizes these cases from its external lexicons, and when such a verb is instantiated, automatically populates the nested proposition with whatever argument the user indicates in the appropriate slot of the controlling verb (and forwards any subsequent changes to that argument). For instance, the verb ordered is a control verb in that the Agent of the subordinate clause is bound to the ixperiencer of the main clause. For instance, the system allows the propositional equivalent of “she ordered him to sit down,” but not “she ordered him for his sister to sit down.”

Using this approach, we have adapted VerbNet into a set of 20,530 predicate frames. However, the mapping from VerbNet to our own slot restrictions is incomplete, in that our system features a coarser and somewhat smaller range of selectional restrictions than what VerbNet offers. We disregard, for instance, VerbNet syntactic frames that deal with values (“the dress cost ten dollars”) or amounts (“he was six feet tall”). This is an implementation detail rather than a design choice. In the future, SCHEHERAZADE can be expanded to more closely adapt the VerbNet system of selectional restrictions, adding support for values, amounts and other concepts, which would increase the expressive range of its approach to propositional modeling.

VerbNet has also been mapped to the large-scale annotation project PropBank [Kingsbury and Palmer, 2002]. There are three key differences between PropBank and this project.
<table>
<thead>
<tr>
<th>VerbNet Thematic Role or Selectional Restriction</th>
<th>VN Syntax Restriction</th>
<th>SCHEHERAZADE Slot Restriction</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theme, Topic, Cause, Stimulus or Communication</td>
<td>that_comp</td>
<td>Stative (assertive)</td>
<td>She told him <em>that he was gracious.</em></td>
</tr>
<tr>
<td>Proposition</td>
<td>that_comp</td>
<td>Stative (imperative) or Timeline (imperative)</td>
<td>She ordered <em>that he be gracious.</em></td>
</tr>
<tr>
<td>Proposition</td>
<td>NOT that_comp or Timeline (imperative)</td>
<td>Stative (imperative, control)</td>
<td>She urged him <em>to be gracious.</em></td>
</tr>
<tr>
<td>Theme, Stimulus or Communication</td>
<td>that_comp</td>
<td>Timeline (assertive)</td>
<td>She told him <em>that the dog had eaten.</em></td>
</tr>
<tr>
<td>Theme, Topic</td>
<td>np_tobe, np_to_inf</td>
<td>Event (infinitive), Stative (infinitive)</td>
<td>She had a desire to <em>swim the English Channel.</em></td>
</tr>
<tr>
<td>Predicate</td>
<td>oc_ing</td>
<td>Stative (gerund, control)</td>
<td>She characterized him as <em>being studious.</em></td>
</tr>
<tr>
<td>Topic</td>
<td>ac_ing</td>
<td>Stative (gerund)</td>
<td>She lectured about <em>being studious.</em></td>
</tr>
<tr>
<td>Theme, Predicate, Topic</td>
<td>sc_to_inf, for_comp</td>
<td>Event (infinitive)</td>
<td>I needed <em>for her to arrive.</em></td>
</tr>
<tr>
<td>Theme, Source, Topic, Proposition</td>
<td>be_sc_ing, oc_ing</td>
<td>Event (gerund, control)</td>
<td>He were forced into <em>using his savings.</em></td>
</tr>
<tr>
<td>Theme, Topic, Proposition</td>
<td>oc_to_inf</td>
<td>Event (control) or Stative (control)</td>
<td>Lack of money forced him <em>to get a job.</em></td>
</tr>
<tr>
<td>Proposition</td>
<td>oc_to_inf</td>
<td>Timeline (imperative)</td>
<td>The accident forced his wife <em>to get a job before April 1.</em></td>
</tr>
<tr>
<td>Theme, Source, Topic, Proposition</td>
<td>sc_ing, np_ing, ac_ing</td>
<td>Event (gerund)</td>
<td>He relied on her <em>arriving before midnight.</em></td>
</tr>
<tr>
<td>Topic</td>
<td>wh_comp</td>
<td>Timeline (whether)</td>
<td>He asked her whether she had <em>broken the lamp.</em></td>
</tr>
<tr>
<td>Any</td>
<td>how_extract</td>
<td>Event (how, instructional), Stative (how, instructional) or Timeline (how, instructional)</td>
<td>I discovered <em>how to do it.</em></td>
</tr>
<tr>
<td>Any</td>
<td>how_extract</td>
<td>Timeline (how, factual)</td>
<td>I discovered <em>how he had done it.</em></td>
</tr>
</tbody>
</table>

Table 4.4: Mappings from VerbNet thematic roles, selectional restrictions and syntactic restrictions to SCHEHERAZADE slot restrictions. Only the mappings for slots restricting to nested propositions and references to alternate timelines are shown.
First, PropBank focuses on the sentence level, where a SIG encoding is a single structure with many interconnected propositions that bind the entire discourse. Second, due to the factors we have discussed, we do not impose a direct mapping from predicates in a textual discourse to propositions in the encoding. In our collection experiments, annotators either altered or consolidated text to focus on underlying story events rather than rhetoric, or skipped over propositional modeling altogether (upon request) to focus on the thematic content found in the SIG’s interpretative layer. Third, PropBank fills its arguments with spans of text, such as assigning a clause to Arg0 or Arg1. Our representation disallows this use; an annotator chooses an element from a formal taxonomy to serve as an argument. For example, rather than highlight the text “the window” to serve as an argument, the annotator would select a noun instantiated from the appropriate WordNet window synset. If no formal symbol can be found for an argument, the annotator “rephrases” the proposition as best as he or she can, and still relates it to the equivalent span of source text.

4.3 Graphical Annotation Interface

To evaluate the accessibility of the SIG model, as well as to collect a corpus of SIG encodings, we ran several collection projects in which we charged subjects with the task of taking a source text and constructing a SIG encoding using Scheherazade. These subjects were undergraduate or graduate students at Columbia who were not experts in either computer science or linguistics (although they included some literature experts). This necessitated the development of a user-friendly graphical interface to make the process of constructing a SIG encoding, including propositional modeling, convenient and enjoyable to such users. This requirement became clear after a formative evaluation we conducted with a baseline approach in which a small set of users drew SIG encodings for a source text using a free-form vector graphics tool and a set of written guidelines. We found that the annotators sometimes drew graphs that were “malformed” with respect to logical constraints of the SIG (such as arc type/node type compatibilities). This motivated us to build a graphical user interface (GUI) that rests atop the Story Logic Module and API and makes the annotation process amenable to an interdisciplinary user community.

The design of such a user interface, one that bridges users to discourse annotation, temporal modeling, theory-of-mind interpretation and propositional modeling, presents challenges. Users must be able to access and instantiate the hundreds of thousands of frames of world knowledge, arrange them on multiple timelines (one for Reality and one for each modal context), be constrained by the logical rules governing interpretative-layer graph topology, and grasp the logical rules of interpretative node actualization. They must not only be able to construct an encoding using a point-and-click interface, but at each step, get feedback that confirms whether or not the system has correctly “understood” the aspect of the story being annotated or encoded. The interface must not only allow users to model stories symbolically, but guide them and provide advice.

In designing the GUI, it also became clear that users would find propositional modeling easier if the workflow featured natural language alone. Propositional form (with predicates, parentheses and lists of arguments) is not easily readable to users who are not highly trained, especially when the propositions are further imbued with temporal and intentional
metadata. Through formative evaluations, we found that the approach with the highest usability was one which hides propositions from the user by means of feedback text—automatically generated textual renderings of the every aspect of the SIG data structure. We built a text generation module to meet this task. Feedback text has allowed us to implement a “natural language in, natural language out” workflow. For example:

1. The user sees an NL phrase in the source text: “A Crow was sitting on a branch of a tree.”

2. The user searches through a list of frames in NL form, until she finds an appropriate frame: “Something sits on something.”

3. The system asks the user to fill in the slots with NL prompts: “What sits?” “On what?” The user answers by selecting from among the instantiated nouns she has set up (the crow, the branch of the tree).

4. The successfully modeled proposition, sits_on(crow1, branch(tree1)), is never revealed to the user. Rather, an NL equivalent is given: “The crow sits on a branch of the tree.” This may or may not be the surface form of the original clause or sentence.

5. The user evaluates the feedback text to check that the concept has been successfully encoded. If the generated feedback text is wrong in some way, the user reselects either the frame or its arguments.

6. The user situates the encoded proposition in a certain state and timeline. The feedback text adjusts the tense and aspect of the feedback text to reflect the temporal positioning of the proposition (Section 4.4): “A crow was sitting on a branch of a tree.”

7. The user repeats the process for other spans of source text. The system juxtaposes proposition-level feedback text to create a continuous discourse called the reconstructed story. The user is able to compare the NL source story to the NL reconstructed story, and when the two are similar to her satisfaction, considers herself finished with timeline layer annotation.

8. A similar “NL in, NL out” approach eases the process of building the interpretative layer of the encoding.

In the remainder of this chapter, we will discuss the interface, annotation process and text generation module which implement this workflow. We will use the Aesop fable “The Fox and the Crow”, shown in Table 4.5, as the subject of a running example. In Chapter 5, we will describe the collection project we have carried out which shows SCHEHERAZADE to be successful at eliciting annotations from trained annotators. We present this work as a contribution.
A Crow was sitting on a branch of a tree with a piece of cheese in her beak when a Fox observed her and set his wits to work to discover some way of getting the cheese.

Coming and standing under the tree he looked up and said, “What a noble bird I see above me! Her beauty is without equal, the hue of her plumage exquisite. If only her voice is as sweet as her looks are fair, she ought without doubt to be Queen of the Birds.”

The Crow was hugely flattered by this, and just to show the Fox that she could sing she gave a loud caw. Down came the cheese, of course, and the Fox, snatching it up, said, “You have a voice, madam, I see: what you want is wits.”

Table 4.5: “The Fox and the Crow”.

4.3.1 Related work

As we mentioned in Chapter 3, previous models of discourse have been accompanied by corpus collection projects which assign the task of semantic annotation to trained annotators. Several of these have dealt with stories in particular. As the SIG model overlaps with several of these projects, so too does our annotation process overlap with prior work.

The marking up of a textual corpus according to a formal model of predicate-argument structure, word sense, thematic role, time, or discourse cohesion is typically done without use of feedback text. When Carlson et al. [2001] applied the Rhetorical Structure Theory model to a large scale collection project, annotators used lexical and syntactic clues to help determine the boundaries between discourse units, then selected the most appropriate RST relations to join units together—a process that required professional language analysts with prior experience in other types of data annotation. The Penn Treebank Corpus [Marcus et al., 1993] collected syntactic and part-of-speech annotations through a combination of automatic tagging and manual correction/tagging by trained annotators. Thematic role (predicate-argument) annotation was applied to the same corpus by the PropBank project [Palmer et al., 2005]; this was also a direct markup of the source text, with users selecting text spans that served as arguments. The more recent Ontonotes project [Pradhan et al., 2007] involves several interconnected layers of annotation, including syntax, propositional structure, word sense disambiguation, named entity classification and anaphoric coreference. The creators use the Penn Treebank and PropBank corpora, adding methods to combine fine-grained WordNet senses to arrive at 90% inter-annotator agreement on word sense disambiguation. Finally, the TimeBank corpus [Pustejovsky et al., 2003b] is annotated to indicate events, times, and temporal relations. After the corpus was pre-processed using automatic tools to find likely temporal anchors, the annotators (who came from a variety of backgrounds) used a modified version of the Alembic Workbench graphical interface to mark up time expressions and connectives (Figure 4.6).

In addition to the textual markup angle, SCHEHERAZADE also lets users browse formal taxonomies of knowledge and instantiate types into instances—not only predicate frames, but nouns themselves. This process has been explored in previous projects with the aim to allow domain experts to assist in the creation of knowledge bases. A typical “knowledge capture” or “knowledge entry” system presents a knowledge-base (KB) editor which
CHAPTER 4. SCHEHERAZADE

Figure 4.6: Prior annotation interfaces: Alembic Workbench (top) and Protégé/OWL.
displays a taxonomy of elements, sometimes in a graphical tree or network, and allows users to add nodes and arcs to introduce new elements and new first-order relations in a point-and-click interface [Paley et al., 1997; Clark et al., 2001]. The most well-known and extensible knowledge capture tool, Protégé, visualizes ontologies of frames (and their instances) through a variety of graphical metaphors [Storey et al., 2001]. A recent extension to Protégé is designed to ease the process of authoring ontologies for the “Semantic Web” using the W3C’s Web Ontology Language (OWL) [Knublauch et al., 2004]. This version of the tool, seen in Figure 4.6, allows users to define classes (frames) with slots, individuals that instantiate the classes, metadata and logical entailment rules.

The use of generated feedback text to make semantic encoding more accessible aligns Scheherazade with the WYSIWYM user-interface pattern: “What you see is what you mean.” Power and Scott [1998] describe WYSIWYM as a key technique for making “symbolic authoring” tools natural, simple and self-documenting. They define a WYSIWYM system as one in which the user only sees feedback text, never the domain model itself, even in the knowledge entry process [Power et al., 1998]. Biller et al. [2005] describe a WYSIWYM editor in which the representation, like ours, is a conceptual graph that incorporates the VerbNet and WordNet linguistic resources. However, compared to this work, our work is more geared toward narrative, and includes more complete representations of time, modality and agency. Power and Evans [2004] explore the effects of varying feedback text (its illocutionary force, time, polarity, modality, and modifiers) to communicate the formal properties of the entity being examined. For example, “the patient took an aspirin” implies a different formal encoding than “the patient may take an aspirin” or the imperative “take an aspirin.”

Projects in the domain of narrative knowledge entry have typically aimed to elicit original stories, rather than annotating existing discourse. Bers [1999], for instance, elicits stories from children by using a programmable plush toy that provides encouraging feedback. Upon parsing the child’s story, a back-end system finds a closely matching story in its database to tell as a reply. Other work has similarly used embodied conversational agents to engage users in conversational storytelling [Bickmore and Cassell, 1999]. As we mentioned in Section 3.2.1, Riedl et. al [2008] recently explored a methodology with which users can author stories in the QUEST formalism and have a system automatically ask relevant feedback questions such as “why did that happen?”.

We see our interface as a contribution not only because it implements interpretative-layer SIG annotation—itself a novel approach to modeling discourse relations in a narrative text—but because of its wide use of the WYSIWYM technique to reflect the system’s understanding of story content in feedback text.

### 4.3.2 Overview of annotation procedure

In the Scheherazade encoding interface, the first task for an annotator is to read the text and fully comprehend it. As a story graph is a discourse unit rather than a sentential unit, it is important for the annotator to take a holistic view of the text before beginning the annotation process. From there, the method for creating an encoding from the source text involves three tasks. The annotator does not need to complete the three tasks sequentially;
one can move back and forth as needed. The tasks are:

1. **Agent, object and theme extraction.** The annotator identifies agents, objects and relevant themes and processes in the text, and represents them as *instance objects* (individuals).

2. **Propositional modeling.** The annotator builds the timeline layer of the encoding. Predicate frames and arguments are selected that best reflect the events, statives and modifiers that appear in the source text. Propositions are assigned to states and transitions on a timeline and linked to corresponding spans of source text. Modal (hypothetical) events are grouped in alternate timelines.

3. **Interpretative modeling.** The annotator builds the interpretative layer of the encoding to model his or her understanding of the overarching goals, plans and beliefs of its agents. Propositions that represent goals, plans and beliefs are modeled and placed in their appropriate agency frames (that is, either as ground truth or inside a belief frame of an agent—see Section 3.3.2). Each node is also annotated in terms of its affectual impact, if any, with respect to each agent.

After making each change to the story graph, the annotator checks the pursuant feedback text to ensure that the system has encoded the concept correctly. The process terminates once the annotator feels that he or she has encoded the story with the greatest amount of precision possible, given the formal limitations of the representation.

Figure 4.7 shows the SCHEHERAZADE GUI. (The two parts of this figure represent panels that are placed side by side in the interface, but we have rearranged them for formatting purposes.) There are three major panels to the interface that correspond to the three major annotation tasks. Minor panels, which can be summoned by the buttons along the bottom-right, allow the user to perform "housekeeping" operations such as saving and loading encodings, undoing or redoing operations, and browsing the annotation guidelines.\(^2\)

The three large buttons along the top of the screen bring the user to the three major panels. The *Story Elements* panel, in Figure 4.7, is used to instantiate objects, creating the story content nodes seen in the "Instantiated Objects" box in Figure 4.2. The *Timelines* panel, seen later in Figure 4.10, is used to model events, statives, and modifiers (to create the timeline content as seen in the "Reality Timeline" area of Figure 4.2). Finally, the *Interpretations* panel, seen later in Figure 4.15, provides a "canvas" on which annotators can encode an agent-centric view of the text (to draw the "Interpretative Nodes" as seen in Figure 4.2).

The smaller panels in Figure 4.7 show the source text and reconstructed story, respectively. Each of the clauses in the source text is highlighted. A highlighted clause is one that has been represented as a textual-layer (TE) node and linked to a timeline-layer node with *interpreted as*. Because all of the clauses are highlighted in this example, we can say that the presented encoding completely covers the source text. In other scenarios, annotators may leave non-narrative spans of the source text, such as news article bylines, un-annotated. Such spans remain in their original, non-highlighted appearance in Source Text panel.

\(^2\)Marshall Fox wrote the component which loads the annotation guidelines from disk.
Figure 4.7: Elements (object extraction) screen of the SCHEHERAZADE interface, including the Story Elements panel (top), and the source/feedback text panels.
4.3.3 Object and theme extraction

After reading the text, the annotator first identifies nouns from among the five classes we enumerated earlier: characters, locations, props, activities and qualities. A noun must be instantiated from a type before it can be invoked as an argument in a proposition. For instance, the first sentence of the fable at hand is “A Crow was sitting on a branch of a tree with a piece of cheese in her beak.” The main event predicate, sitting, has two thematic roles, an agent and a destination. The agent is a crow, and the destination is the branch of a tree. We must create instance objects for these two nouns so that we can invoke them as arguments when instantiating the sit predicate frame. The latter clause in the source text, about the piece of cheese, is a positional stative which we can model separately once we establish a piece of cheese and the crow’s beak as objects.

This system is more complex than the PropBank approach of highlighting text spans in the source text to serve as arguments. The major advantage to object extraction is that instance objects are reusable as arguments—they function as typed entities with coreferent mentions. By creating and reusing instance nodes, annotators perform coherence resolution as part of the encoding process. As we saw in Chapter 3, coreference is key to an encoding that shows the cohesion of a discourse.

Figure 4.8 shows the process in the GUI for creating a new instance object for the Fox in our fable:

1. The annotator first selects the object class by clicking the appropriate tab on the top of the panel (character, location, prop, quality or behavior). Since we are creating an instance node for the fox, we select the Character tab.

2. The left side of the panel shows a list of instance objects that have already been created. The system allows for instance objects which are themselves abstractions within the story-world, such as the “wits” that the fox wishes upon the crow. Individuals that are concrete within the story-world are listed below abstractions.

3. The right side of the panel shows an empty form with a type selector. The selector is populated with the tens of thousands of types that we imported from WordNet, arranged hierarchically according to WordNet’s hyponymy tree. A search box allows the user to type in a string and see a list of matching types, including the various types of foxes distinguished in WordNet. Immediate hyponyms are used in parentheses to allow the user to quickly disambiguate between noun senses (e.g., fox (canine)).

4. Once the user selects a type, the form prompts for certain metadata. An accept button shows generated feedback text; when the annotator clicks the button, the new character becomes fully instantiated and appears in the left-side list. The annotator can provide a gender and a name, which the feedback text generator uses to select appropriate pronouns and references, respectively.

Previous work in word sense disambiguation has shown WordNet synsets to be very fine-grained, which can hurt inter-annotator agreement [Palmer et al., 2007]. For example, “fox” also represents the synset of “a shifty deceptive person;” other distinctions can be
Figure 4.8: Instantiating an object in the SCHEHERAZADE Story Elements screen. Selecting
an object type (top), and supplying metadata.
far more subtle. We address this issue in two ways. First, at the user interface level we show disambiguators for each type. The annotator sees “fox, canine, carnivore” juxtaposed with “fox, deceiver, wrongdoer.” The second technique is to limit ambiguities by selectively disassociating certain lexemes from their less-used synsets. Specifically, we set a threshold for the information content a synset must have to serve as an option for an instance object. WordNet provides this attribute from a model of the usage of each synset in a naturally occurring corpus.

Noun phrases that include part-of and attributional relationships can be instantiated with two special frames: \textit{Part of something} provides for non-exclusive part-of relationships (“a branch of the tree”) while \textit{Part of something (unique)} refers to exclusive relationships (“the beak of the crow”) or attributed qualities (“the hue of the bird’s plumage”). Figure 4.9 shows the process of creating an instance object for the noun phrase “a branch of a tree.” At the top, the annotator selects the \textit{Part of something} frame at the top of a list of prop types. The form is populated with two slots that need filling: the type of part and the object to which the part is attributed. Each slot is accompanied by a red \textbf{question button} which the annotator selects when she is ready to answer the question and fill the slot. Upon clicking the first question button, “What type of part?”, the user is prompted to select a type from the searchable type selector—in this case, \textit{branch}. The accept button remains flat, indicating that the form needs additional slots to be filled before the object can be instantiated, but the feedback text on the face of the button expresses the partially completed object: “A branch of something.” When the annotator clicks the second question button, “Part of what?”, the selector presents a list of existing instance objects from which to choose. Upon picking the appropriate host for the branch, \textit{the tree}, the accept button becomes raised and clickable, reading “A branch of a tree.” Clicking the raised accept button prompts the system to create the instance object, which can in turn be used as a “host” object for another part (\textit{the edge of the branch of the tree}).

The object extraction panel also allows the annotator to provide \textbf{attributes} to each instance object. Attributes are statives which are immutable throughout the story-world. If a character is known to be tall, beautiful, or jealous, such a static can be assigned as an attribute of the character. The feedback text generator uses the attribute when introducing characters: “A beautiful maiden.”

### 4.3.4 Propositional modeling

The annotator constructs the timeline layer of the SIG encoding by carefully instantiating events, statives and modifiers, and assigning them to points in a main Reality timeline or an alternate modal timeline (an approach we introduced in Section 3.3.1). From a user interface standpoint, this part of the annotation process takes place in the Timelines panel of the GUI. The panel features a visualization of a timeline as a vector of boxes—“Story Points” stand in for State nodes. When the annotator selects a Story Point, it slides to the

---

\(^3\)The relationship between Story Points at the GUI level and State nodes in the SIG is indirect. Each Story Point refers to the span of time between two adjacent State nodes. While an event cannot occur in a single instant of time, it can occur in a single Story Point. We inserted this layer of abstraction after formative evaluations showed the state/span distinction to be confusing to users.
Figure 4.9: Noun phrases can be instantiated with object frames.
center of the panel, and all instance events (instantiated verb predicates) and instance statives which occur during that Story Point are listed below the timeline (whether or not they start or end during that Story Point).

To create a new instance event or instance stative, the annotator first navigates to the Story Point where the event or stative is to begin. She then selects “New Action” or “New Property,” respectively, from the top of the panel. The main section of the panel then presents a form and a selector very similar to those used in the object extraction panel. The process is similar as well: First, select a type (in this case, a verb or stative predicate); second, fill in appropriate slots with arguments as prompted by question buttons; third, verify the results using the feedback text on the accept button, making changes as necessary; and fourth, click the accept button to create the new node.

Figure 4.11 illustrates this process with respect to the sits clause from the first sentence of “The Fox and the Crow”: “A Crow was sitting on a branch of a tree.” First, the annotator searches for an appropriate predicate frame by typing the present-tense form of a verb into the search panel of the type selector. The selector has indexed all of the “menu names” which we previously generated in the process of adapting VerbNet records (Figure 4.5). In this case, a search for “sits” returns seven frames which differ in their thematic roles (slots) and their use of prepositions. Based on the syntactic construction of the sentence she is attempting to model, the annotator selects the Something sits away from/near/on/toward something frame. The system populates the form with three questions: Who or what is sitting, who or what is the destination, and what is the preposition? The annotator clicks each question button in turn. For both the “who or what” questions, the annotator selects from among the instance objects she previously modeled: the crow is the agent, and the branch of the tree is the destination. The annotator then selects “on” as the appropriate preposition, and checks the feedback text that has been generated and displayed in the

---

4We altered the terminology for usability reasons following formative evaluations.
Figure 4.11: Modeling an instance stative from the SCHEHERAZADE Timelines screen.
accept button: “The crow sits on the branch of the tree.” The annotator also has the option to negate the proposition (“the crow does not sit on the branch of the tree”), but in this case, simply clicks the accept button to construct the instance stative and attach it to the timeline at the Story Point which she previously indicated.

Before clicking the accept button on any new event or stative, the user highlights the span of source text that corresponds to the new content. This has the effect of creating a new Text (TE) node in the textual layer of the SIG. The propositional content is attached to the new Event or Stative node with an interpreted as arc, as in Figure 3.7. A single span can be associated with multiple propositions. Note that the order of highlighted text spans in the Original Story box need not match the order of propositions on the timeline; this allows annotators to capture differences between the telling time and the story time (Section 3.3).

Once constructed, the new instance event or stative appears below the timeline, as in Figure 4.10, in the form of feedback text; the overall feedback text for the story as a whole (in the Reconstructed Story panel) is updated to reflect the new content. The annotator can then edit or delete previously constructed content. Specifically, button near each span of feedback text allow users to attach modifiers (constructed as propositions, as in Figure 4.11), change the predicate frame, fill the frame with different arguments, reassign the event or stative to a different beginning or ending Story Point, or delete the story content altogether.

The argument selection area of the form changes with respect to the thematic role of the slot being filled, according to the VerbNet mappings we described in Tables 4.3 and 4.4. For an Experiencer slot, the interface presents a list of extracted characters; for a Communication slot, it prompts the user to select a predicate frame to serve as the basis for a nested dialogue proposition. Figure 4.12 shows an example of the process for nesting propositions. Upon selecting the communicative frame Something says some proposition, the annotator can select a predicate frame for the dialogue act from among those permitted by the encapsulating frame’s selectional restrictions (“how” action, alternate timeline, and so on). A second form, color-coded differently, is graphically nested inside the first, and the annotator proceeds to fill out the inner form. (Forms can be nested indefinitely to create complex propositions.) Once the inner proposition is satisfactorily complete, the accept button shows overall feedback text such as “The fox says that the crow is noble.”

The sentence we have modeled, about the crow sitting on the branch, is straightforward. At other times, a fair amount of simplification of the source text is necessary. For example, our guidelines have annotators phrase passive voice statements as active voice where possible. For “The crow was hugely flattered by this,” the annotator would encode the equivalent of “The fox flattered the crow” or “the fox’s words flattered the crow.” Note that the word “flatter” in this case, itself a translation of the original Greek, carries certain semantic ambiguities relating to the theory-of-mind interpretation of the text. Did the fox have an ulterior motive or did he simply embarrass the crow with excessive praise? Did the crow doubt the fox’s sincerity or believe that he was being genuine? We ask annotators to use interpretative-layer relations to answer such questions whenever possible, rather than try to disambiguate the original text with complex propositional constructions. When this is not possible, annotators replace idioms and figures of speech with their closest approxi-
Figure 4.12: Complex propositions can be modeled by nesting an event, stative or alternate timeline as an argument.
Alternate timelines, which we introduce in Section 3.3.1, are useful for referring to past events and statives, possible futures, hypothetical scenarios and other modal content. An alternate timeline is attached to a “parent” timeline with an equivalent arc that runs between those states that act as a common point of reference between the two frames of time. Annotators can easily create an attach timelines using the graphical interface. A dropdown menu in the Timelines panel lets users switch from the Reality timeline to an alternate timeline, and to create a new timeline with a particular parent timeline. The view of an alternate timeline is similar to that of the Reality timeline, except the color scheme is different, and the Reconstructed Story box only displays the feedback text equivalent of the alternate timeline (as opposed to a rendering of the entire story).

One situation where alternate timelines are put to use in “The Fox and the Crow” is when the fox makes a devious offer to the crow in saying, “If only her voice is as sweet as her looks are fair, she ought without doubt to be Queen of the Birds.” The fox is not referring to an actual event, but a hypothetical event, and the consequences which would follow. We set up an alternate timeline called “Fox offer” that features two statives, one in which the sweetness of the crow’s voice is equal to the fairness of the crow’s looks, and one in which the crow is the Queen of the Birds (Figure 4.13). In the interpretative layer of the SIG, we model the same if-then relationship between the two statives with a would cause arc inside a belief frame of the crow, which is itself within a goal frame of the fox—since the fox wants the crow to believe that one would lead to the other (a deception pattern, as in Figure B.10 in Appendix B). Propositional modeling in the timeline layer offers a parallel mechanism for expressing conditionals and subjunctives. Each event and stative can be marked with a flag called \textit{IF} that indicates conditionality: if \textit{IF} is invoked in an alternate timeline, whatever content that is not marked with \textit{IF} is interpreted as being predicated on the \textit{IF} content.
seen in Figure 4.13, the annotator marks only the voice-looks equivalence with *IF*, because the identity of the crow as Queen of the Birds is the consequence rather than the condition.

Alternate timelines are visualized with a button named *Attachment Point* below the currently selected State Time box. This button determines which State node is incident to the *equivalence* arc connecting the alternate timeline to its parent. The other end of the *equivalence* arc is attached to the event or stative that invokes the alternate timeline, such as in Figure 4.14. The selection of attachment point determines what parts of the alternate timeline are past, present and future with respect to the parent timeline. If the attachment point for “Fox offer” were at an earlier Story Point than either the “IF” clause or the “THEN” clause, then the alternate timeline would be a possible future, predicated on the crow’s voice becoming as sweet as her looks are fair.

We noted in Sections 3.4 and 4.2 that propositional modeling is “optional” with respect to the formal definition of the SIG schemata. Proposition and Interpretative Proposition nodes need only be annotated with the identities of their agents. At the user interface level, this is simply a matter of choosing an agent to act and using it in generic predicate frame such as *Something acts*. The annotator still highlights the span of source text she wishes to represent, but the timeline node itself only includes the identity of the acting agent (if any). The timeline node can then be connected to the interpretative-layer content. This usage pattern reduces the semantic precision offered by fuller propositional encoding, but also reduces the corresponding issues of coverage. The distinction between “story time” and “telling time” is preserved, as is the relationship between the surface form a discourse (the source text) and the rich set of discourse relations available in interpretative-layer annotation.

### 4.3.5 Interpretative panel

The third and final panel of the SCHEHERAZADE GUI, Interpretations, presents a canvas for annotators to draw nodes and arcs that represent their interpretative-layer (theory of mind) modeling of the source text. Annotators have a more direct control over the encoding’s graph structure here than in the previous panels, in that they can view and manipulate nodes and
their incident arcs rather than use abstractions such as Story Points.

Figure 4.15 displays this panel with respect to a completed annotation for “The Fox and the Crow”. The large expanse of the panel contains three columns of boxes. The first two columns of boxes are automatically populated onto the canvas as the annotator builds the timeline: The rightmost column contains blue boxes that represent the content of the Reality timeline, chronologically ordered and “flattened” into a single vector (modifiers are offset slightly); the leftmost column contains the spans of the source text that the annotator associated with each proposition. The remainder of the panel, to the right of the second column, begins as a blank canvas. The annotator is charged with filling it with interpretative content according to the guidelines we outlined in Section 3.3.2. In a sense, this panel offers the annotator a view of the entire encoding in graph form, including the textual, timeline and interpretative layers.

Interpretative content includes agency frames (goals, plans and beliefs), Interpretative Proposition nodes and Affect nodes. These are drawn as boxes with purple labels, tan-colored labels, and black labels, respectively. The annotator can construct new nodes using the “Create” button at the top of the panel and then drag them around the canvas until they are positioned to her satisfaction. If the annotator wishes to model a proposition for an I node, the interface summons the proposition construction panel, with the same frame selector and argument-specification form as in the other two interface panels. New nodes can be created inside an existing agency frame or in ground truth (the white background canvas), a distinction whose semantic entailments we discussed in Section 3.3.2.

The Reconstructed Story panel on the lower-right corner has been replaced here with an “Interpretative Detail” panel. Whenever the user selects a node or a frame, the panel provides three features:

1. Feedback text which has been generated to describe the selected node or frame. In the case of a frame, the feedback text summarizes the goal, plan or belief inside the frame.

2. A **new arc** button allows the annotator to draw a new interpretative arc from the selected node to another node. A dropdown panel shows the arc types which can legally originate from the selected node, according to the rules of the SIG schemata.

3. A list of the nodes which are connected to the selected node. The arc types that describe the connection are also displayed. The annotator can click on the adjacent node to select it, or delete the incident arc.

The bottom-right corner of Figure 4.15 shows the interpretative detail box that is displayed for the goal frame representing the fox’s ultimate goal to obtain the cheese. The automatically generated feedback text describes the goal, as well as the affectual impact determined by the connected Affect nodes (positive for the fox’s ego and the fox’s health). Two timeline nodes are connected to this frame, and by extension, two spans of source text are connected as well: “The fox set his wits to work to discover some way of getting the cheese” is interpreted as this goal box, and “the fox snatches the cheese” actualizes it.

We saw in Section 3.3.2.1 that each node and frame of interpretative content carries an actualization status that is logically entailed for each point in story time (each state in the
Figure 4.15: Interpretative annotation panel.
CHAPTER 4. SCHEHERAZADE

Reality timeline). Each node begins in a Hypothetical (H) status, and then becomes either Actualized (A) or Prevented/ Ceased (PC) if a timeline-layer node connects to it with an actualizing arc (interpreted as, implies, actualizes) or the ceasing arc. As the actualization status of a node determines whether it is effectively true at each point in story time, the annotator must carefully track the actualization triggers that she inserts with these four arcs. To aid in this process, the annotation tool calculates and displays the actualization status for each interpretative node when a timeline node is selected.

Figure 4.16 gives an example of actualization status highlighting. The top panel shows the section of the interpretative graph for “The Fox and the Crow” which models the fox’s goal to obtain the cheese. The fox plans to flatter the crow, which would cause the crow to plan to sing in order to prove its worth, which would cause the crow to open its beak and drop the cheese. As no node is selected in this panel, the nodes and boxes are colored as usual. The bottom panel of Figure 4.16 shows the effects of selecting the timeline stative “The crow feels that the fox has flattered her.” The plan node for “The fox flatters the crow” is now shaded with a dark green, indicating that it has been actualized (indeed, the timeline stative connects to this node with an actualizes arc). The label of the goal box (THE FOX: GOAL) is also shaded with dark green, since the fox’s plan frame itself is long since actualized at this point in the story. All the other nodes are grey because they are still hypothetical—we do not yet know if the fox will succeed in prompting the crow to devise a plan to sing, or if the crow will open its beak. However, the nested goal box (the crow’s potential plan) is shaded with a light green, indicating that the fox expects the frame to be actualized (because he believes the successful flattering would cause it). We describe expectations in more detail in Section B.1.

4.3.6 Conclusion

This section has given a walkthrough of the highlights of the graphical user interface we have written on top of the SCHEHERAZADE API to make the annotation process amenable to community annotation efforts. Given a source text as input, annotators can construct nodes to represent named entities, use dynamic forms to fill out the slots of event and stative frames, and encode the meaning of the story with interpretative-layer content.

We built this interface in iterations. Three separate formative evaluations, each involving compensated users from outside our department, guided its development by attempting to encode a set of Aesop’s fables. These users gave valuable feedback regarding the system’s ease of use, the coverage of its knowledge base, and the expressiveness of the discourse relations. We followed up with two “production” collection experiments which we describe in Chapter 5. First, however, let us discuss in some detail the text generation module which provides the WYSIWYM feedback text. This generated text proved essential to our efforts to improve the system’s ease of use.

4.4 Text Generation: Assigning Tense and Aspect

While SCHEHERAZADE is not strictly a project in text generation, we found through formative evaluations that generated feedback text greatly enhances the usability of the interface.
and standing
under the tree

he looked up

and said, "What a noble bird I see above me!
Her beauty is without equal
the fox says that he sees a bird
the fox says that the beauty of the crow is unique
the hue of her plumage exquisite
the fox says that the hue of the plumage of the crow is exquisite
If only her voice is as sweet as her looks are fair, she ought without doubt to be Queen of the Birds."
The Crow was hugely flattered by this

she gave a loud caw
the crow caws

the fox stands under the tree
the fox looks toward the crow
the fox says that he sees a bird
the fox says that the beauty of the crow is unique
the hue of her plumage exquisite
the fox says that the hue of the plumage of the crow is exquisite
The Fox's Goal
the fox persuades the crow to open the beak of the crow
the crow opens the beak of the crow
the crow is able to sing

The Fox's Goal
the fox persuades the crow to open the beak of the crow
the crow is able to sing

Figure 4.16: Normal interpretative graph interface (top) and interface with elements color-coded for actualization status relative to a timeline proposition.
Our text generation module reverses the process of semantic encoding by taking a portion of the encoding as a starting point and applying a model of syntax, tense, and aspect to render and display a natural-language equivalent in English. We demonstrated in the previous section how feedback text is used in every facet of the user interface, including renderings of both small pieces of the data structure (e.g., noun types) and large ones (the entire timeline layer as a “Reconstructed Story”). The SCHEHERAZADE API also provide access to the generation module, allowing third-party tools to repurpose feedback text outside of the GUI. This section provides an overview of the generation algorithm we have implemented, and goes into detail on the question of assigning tense and aspect when rendering events that are situated in a formal encoding of time. This work on tense and aspect is the most novel aspect of our approach.

4.4.1 Basic Planner and Realizer

The module is best summarized as a small series of rules which can be called within a generation command. Most rules are tied to a particular encoding facet that one would wish to serialize as text. For instance, a \texttt{generateCharacter} rule will return a textual rendering of an instance character (\texttt{crow} becomes “a crow”). Upon execution, each rule carries out three tasks:

1. Constructs a \textit{generation plan}, a series of instructions that call other rules upon execution;
2. Updates keys and values in a state object; and
3. Issues a sub-command to execute its generation plan, calling each of the rules in its plan in sequence and passing them the state object.

Some rules directly \texttt{emit} a particular lexeme or symbol to a \textit{serializer}. The overall result is a tree structure of hierarchical plans; the tree is generated “on the fly” during the course of a preorder traversal. In a sense, this process acts as a grammar; a simplified list of the generation rules and their plans (involving other rules) is seen in Table 4.6. Unlike in a context-free grammar, though, a state object allows each node in the tree (that is, each rule in execution) a common “whiteboard” with which to communicate.
Figure 4.17: Progressive construction and traversal (execution) of a tree of generation rules. Each rule emits appropriate lexemes and updates/consults a state object.

An example of this process is shown in Figure 4.17. We wish to generate text from the proposition \texttt{walk(john)}, an event on the Reality timeline, from the temporal perspective of a point in time between the proposition’s begin and end times. In 4.17(i), a \texttt{generateEvent} rule takes the proposition as input, along with a constant, \texttt{PRESENT_TELLING}, that expresses our temporal perspective that the event is presently occurring. The rule also uses the world knowledge module—in particular, the generation frame for the \texttt{walk()} verb we earlier adapted from a VerbNet record (Figure 4.5 in Section 4.2). The knowledge base tells us that to serialize the event, \texttt{generateEvent} must call two other rules: one to render the agent who is walking, and one to render the \texttt{walk} verb.

In 4.17(ii), the algorithm recurses on the first call, \texttt{generateCharacter(john)}. Throughout this figure, the rounded rectangle that is shaded is the one currently being executed. The \texttt{generateCharacter} rule checks the state object for whether the \texttt{john} instance object is \emph{given} or \emph{new} to the discourse, and finding it to be new, dynamically inserts a call to a rule
called *introduceCharacter* before a call to emit the name “John.” It also updates the state object to record that John has been introduced. In 4.17(iii), the *introduceCharacter* rule eventually emits three words, “a man named,” though for brevity we do not show a nested call to a rule *generateType* for rendering “man.” (We also do not show the recursive call to a *generateModifier* rule, which emits text when modifiers are associated with events.) 4.17(iv) shows the completion of the tree traversal, in which *generateVerb* uses a model of tense and aspect to determine that “walks” is the correct form of the verb to emit.

Once the evaluation of the plan generation tree is complete, the emitted lexemes are collected and post-processed by the serializer. This function joins together clauses, sentences, and paragraphs with appropriate punctuation, spacing and capitalization: “A man named John walks.” It also maintains an index which maps each clause to the semantic structure from which the clause was derived. This allows the GUI to provide a feature whereby the user can click on a word in the Reconstructed Story panel and navigate to the corresponding point in the story in any other panel—the Timelines panel selects the appropriate Story State, the Source Text panel highlights the equivalent span of source text, and the Interpretations panel selects the node as it appears in the canvas view (Figure 4.18).

To generate a full discourse from the story’s timeline, we implemented several higher-level rules. A *generateTimeSpan* rule concatenates all events that begin or end during that span in a single list, aggregating by the common agent (“X did Y and Z”); a *generateStory* rule renders the entire story by “scrubbing” the timeline from the beginning to the end and generating a sentence or paragraph for each relevant time span. Note that the feedback
text in the Reconstructed Story panel is therefore generated with a strictly linear telling of story content, even if the source text has temporal disfluencies (Figure 3.8 in Section 3.3.1).

We took special care with the generation of referring expressions. Entities given proper names (by annotators) are introduced as such when they are new (“a man named John”), but subsequently referred to by name alone (“John”) rather than nominal (“the man”). When multiple unnamed entities are used, we use ordinal disambiguators (“a rooster fought a second rooster; the first rooster defeated the second rooster”). Discrete and continuous objects are given different determiners (“a” vs. “some”). We use the state object to determine which entity was the most recent to be mentioned for a particular gender, and if possible, replace a name with the pronoun appropriate to the thematic role (e.g., “she walked to him,” “it destroyed itself.”) We also use the state object in conjunction with the control-verb metadata to determine when an agent in a subordinate clause can be assumed and not stated. For instance, Figure 4.19 shows the generation plan tree and resulting serialization for a timeline with a single event:

\[ \text{ask}(\text{jill}, \text{john}, \text{driveTo}(\text{john}, \text{jill}, \text{store})) \]

Because \text{ask} is a control verb, binding the entity being asked, we do not include “John” in the subordinate clause. Leaving aside character introductions, the result is “Jill asked John to drive her to the store,” rather than “Jill asked John for John to drive her to the store.” For metadata on verbs and nouns, including control, irregular conjugations, irregular plural forms, and discreteness, we relied on the COMLEX lexicon [Macleod et al., 1994].

For the remainder of this section, we focus on the assignment of tense and aspect by the \text{generateStory} and \text{generateTimeSpan} rules. All generation systems that communicate knowledge about time must select tense and aspect carefully in their surface realizations. An incorrect assignment can give the erroneous impression that a continuous action has ended, or that a previous state is the current reality. Correct assignments are particularly important in the generation of narrative discourse, where statives and actions occur over connected intervals.

We will describe two contributions: first, a general application of theories of tense, aspect and interval logic to a generation context in which we map temporal relationships to specific tense/aspect selections. Second, an implementation of this approach in the basic sentence planner and realizer we have now described as the \text{SCHEHERAZADE} textual generation module. The first result does not depend on the second.

The discussion is organized as follows: After discussing related work in Section 4.4.2 and reviewing our formal model of time in Section 4.4.3, we describe our method for selecting tense and aspect for single events in Section 4.4.4. Section 4.4.6 follows with more complex cases involving multiple events and shifts in temporal focus (in particular, direct speech, indirect speech, modals and conditional events).

### 4.4.2 Related Work

There has been intense interest in the interpretation of tense and aspect into a formal understanding of the ordering and duration of events. This work has been in both linguistics [Smith, 1978; Dowty, 1979; Nerbonne, 1986; Vlach, 1993] and natural language understanding. Early systems investigated rule-based approaches to parsing the durations and
Figure 4.19: The preorder traversal of a generation plan tree involving a subordinate clause.
orderings of events from the tenses and aspects of their verbs [Harper and Charniak, 1986; Hinrichs, 1987; Webber, 1987; Song and Cohen, 1988; Passonneau, 1988]. Allen [1984] and Steedman [1995] focus on distinguishing between achievements (when an event culminates in a result, such as “John builds a house”) and processes (such as walking). More recent work has centered on markup languages for complex temporal information [Mani, 2004] and corpus-based (statistical) models for predicting temporal relationships in unseen text [Mani et al., 2006; Lapata and Lascarides, 2006].

Our annotation interface requires a fast realizer that can be easily integrated into an interactive, online encoding tool. We found that developing a custom realizer as a module to our Java-based system was preferable to integrating a large, general purpose system such as KPML/Nigel [Mann, 1983; Matthiessen and Bateman, 1991] or FUF/SURGE [Elhadad and Robin, 1996]. These realizers, along with RealPro [Lavoie and Rambow, 1997], accept tense as a parameter, but do not calculate it from a semantic representation of overlapping time intervals such as ours (though the Nigel grammar can calculate tense from speech, event, and reference time orderings, discussed below). The statistically trained FERGUS [Chen et al., 2002] contrasts with our rule-based approach.

Dorr and Gaasterland [1995; 2002] and Grote [1998] focus on generating temporal connectives, such as “before,” based on the relative times and durations of two events; Gagnon and Lapalme [1996] focus on temporal adverbials (e.g., when to insert a known time of day for an event). By comparison, we extend our approach to cover direct/indirect speech and the subjunctive/conditional forms, which they do not report implementing. While our work focuses on English, Yang and Bateman [2009] describe a recent system for generating Chinese aspect expressions based on a time interval representation, using KPML as their surface realizer. For an English-language grammar tutoring system, Fum et al. [1991] introduce extra-linguistic entity (what they call an “objective tense”) that reflects the temporal semantics of the clause or sentence being rendered, then map this entity onto a grammatical tense; we take a similar two-step approach.

Several other generation projects also involve encodings of narrative discourse. Callaway and Lester’s STORYBOOK [2002] aims to improve fluency and discourse cohesion in realizing formally encoded narratives; Ligozat and Zock [1992] allow users to interactively construct sentences in various temporal scenarios through a graphical interface.

### 4.4.3 Temporal knowledge

The propositions that we aim to realize take the form of a predicate, one or more arguments, zero or more attached modifiers (either a negation operator or an adverbial, which is itself a proposition), and an assignment in time. Each argument is associated with a semantic role (such as Agent or Experiencer), and may include entities backed by nouns (such as characters) or other propositions. Predicates include both durative actions and statives [Dowty, 1979]; we will refer to both as events as they occur over intervals. For example, here are two events:

\[
\text{walk(Mary,store,2,6)} \tag{4.1}
\]
hungry(Julia, 1, ∞) \tag{4.2}

The latter two arguments in (4.1) refer to time states in a totally ordered sequence; Mary starts walking to the store at state 2 (as encoded by the ba arc) and finishes walking at state 6 (via \( ea \)). (4.2) begins at state 1, but is unbounded (Julia never ceases being hungry). We do not currently address the use of reference times (such as equating a state to 6:00 or “yesterday”).

(4.1) and (4.2), depending on the situation, can be realized in several aspects and tenses. We adapt and extend Reichenbach’s [1947] system of symbols for distinguishing between simple and progressive aspect. Reichenbach identifies three points that define the temporal position of the event: the event time \( E \), the speech time \( S \) and a reference time \( R \) which may or may not be indicated by a temporal adverbial. The total ordering between these times dictates the appropriate tense and aspect. For example, the simple past “John laughed” has the relation \( E < S \), \( R = E \) because there is no separate reference time involved. The past perfect “John had laughed [by the end of the play]” has the relation \( E < R < S \), in that it describe “the past of the past,” with the nearer “past” being \( R \) (the end of the play). \( R \) can be seen as the temporal focus of the sentence.

As Reichenbach does not address events with intervals, we redefine \( E \) as the tuple describing the onset and termination states attached to the event (for example, (2,6) for Mary’s walk). This definition deliberately assumes that no event ever occurs over a single “instant” of time. The perception of an instantaneous event, when it is needed, is instead created by dilating \( R \) into an interval large enough to contain the entire event, as in Dowty’s approach [Dowty, 1979].

We also distinguish between two generation modes: realizing the story as a complete discourse (narration mode, as seen in the “Reconstructed Story” panel) and describing the content of a single state or interval (snapshot mode, as seen in the Timelines and Interpretations panels). Our system supports both modes differently. In narration mode, we realize the story as if all events occur before the speech time \( S \), which is the style of most literary fiction. (We shall see that this does not preclude the use of the future tense.) In snapshot mode, speech time is concurrent with reference time so that the same events are realized as though they are happening “now.” The system uses this mode to allow annotators to inspect and edit what occurs at any point in the story. In Figure 4.18, for instance, the fox’s observing of the crow is realized as both a present event in snapshot mode (“the fox observes the crow”) and narrated as a past event (“a fox observed the crow”). In both cases, we aim to precisely translate the propositions and their temporal relationships into reliable feedback text. In the remainder of this section, we describe our method for assigning tenses and aspects.

4.4.4 Expressing single events from a reference state

In both snapshot and narration modes, we often need to render the events that occur at some reference state \( R \). We would like to know, for instance, what is happening now, or what happened at 6:00 yesterday evening. The tense and aspect depend on the perspective of the reference state on the event, which can be bounded or unbounded. The two-step process for this scenario is to determine the correct perspective, then pick the tense and
We define the set of possible perspectives to follow Allen [1983], who describes seven relationships between two intervals: before/after, meets/met by, starts/started by, during/contains, finishes/finished by, and equals. Not all of these map to a relationship between a single reference point and an event interval. Table 4.7 maps each possible interaction between $E$ and $R$ to a perspective, for both bounded and unbounded events, including the defining relationships for each interaction. A diamond for $E_1$ indicates at or before, i.e., the event is either anteriorly unbounded ($E_1 = -\infty$) or beginning at a state prior to $R$ and $E_2$. Similarly, a diamond for $E_2$ indicates at or after.

Once the perspective is determined, covering Reichenbach’s $E$ and $R$, speech time $S$ is determined by the generation mode. Following the guidelines of Reichenbach and Dowty, we then assign a tense for each perspective/speech time permutation in Table 4.8. Not all permutations map to actual English tenses. Narration mode is shown as *Future Speech*, in that $S$ is in the future with respect to all events in the timeline. (This is the case even if $E$ is unbounded, with $E_2 = \infty$.) Snapshot mode is realized as *Present Speech*, in that $R = S$. The fourth column indicates the syntactic construction with which our system realizes the permutation. Each is a sequence of tokens that are either cue words (*began*, *stopped*, etc.) or conjugations of the predicate’s verb. “Posterior” is how Reichenbach refers to the “future-of-a-past” situation, for which no tense exists in English; we use the “was going” construction as in “Mary was going to [later] walk to the store.” These constructions emphasize precision over fluency.

As we have noted, theorists have distinguished between *statives* that are descriptive (“John was hungry”), *achievement* actions that culminate in a state change (“John built

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Relations</th>
<th>Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>$R &lt; E_1$</td>
<td>Before</td>
</tr>
<tr>
<td><img src="image2" alt="Diagram" /></td>
<td>$R = E_1$</td>
<td>Begin</td>
</tr>
<tr>
<td><img src="image3" alt="Diagram" /></td>
<td>$E_1 &lt; R$</td>
<td>During</td>
</tr>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td>$R = E_2$</td>
<td>Finish</td>
</tr>
<tr>
<td><img src="image5" alt="Diagram" /></td>
<td>$R &gt; E_2$</td>
<td>After</td>
</tr>
</tbody>
</table>

Table 4.7: Perspective assignment for viewing an event from a reference state.
the house”), and activities that are more continuous and divisible (“John read a book for an hour”) [Dowty, 1979]. Prior work in temporal connectives has taken advantage of lexical information to determine the correct situation and assign aspect appropriately [Moens and Steedman, 1988; Dorr and Gaasterland, 1995; Gagnon et al., 2006]. In our case, we only distinguish between actions and statives, based on information from WordNet and VerbNet. We use a separate table for statives; it is similar to Table 4.8, except the constructions replace verb conjugations with insertions of be, been, being, was, were, felt, and so on (with the latter applying to affective states). We do not currently distinguish between achievements and activities in selecting tense and aspect, except that the annotator is tasked with “manually” indicating a new state when an event culminates in one (e.g., “The house was complete”). Recognizing an achievement action can benefit lexical choice (better to say “John finished building the house” than “John stopped”) and content selection for the discourse as a whole (the house’s completion is implied by “finished” but not by “stopped”).

To continue our running examples, suppose propositions (4.1) and (4.2) were viewed as a snapshot from state $R = 2$. Table 4.7 indicates Begin to be the perspective for (1), since $E_1 = R$, and Table 4.8 calls for a tense/aspect permutation that means “begins at the present time.” When the appropriate construction is inserted into the overall syntax

<table>
<thead>
<tr>
<th>Perspective</th>
<th>Generation mode</th>
<th>English tense</th>
<th>System’s construction</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>After</td>
<td>Future Speech</td>
<td>Past perfect</td>
<td>had {PAST PART.}</td>
<td>She had walked.</td>
</tr>
<tr>
<td></td>
<td>Present Speech</td>
<td>Present perfect</td>
<td>has/have {PAST PART.}</td>
<td>She has walked.</td>
</tr>
<tr>
<td></td>
<td>Past Speech</td>
<td>Future perfect</td>
<td>will have {PAST PART.}</td>
<td>She will have walked.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>to have {PAST PART.}</td>
<td>to have {PAST PART.}</td>
<td>To have walked.</td>
</tr>
<tr>
<td>Finish</td>
<td>Future Speech</td>
<td>“Finished”</td>
<td>stopped {PRP.}</td>
<td>She stopped walking.</td>
</tr>
<tr>
<td></td>
<td>Present Speech</td>
<td>“Finishes”</td>
<td>stops {PRP.}</td>
<td>She stops walking.</td>
</tr>
<tr>
<td></td>
<td>Past Speech</td>
<td>“Will finish”</td>
<td>will stop {PRP.}</td>
<td>She will stop walking.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>to stop {PRP.}</td>
<td>to stop {PRP.}</td>
<td>To stop walking.</td>
</tr>
<tr>
<td>During</td>
<td>Future Speech</td>
<td>Past progressive</td>
<td>was/were {PRP.}</td>
<td>She was walking.</td>
</tr>
<tr>
<td></td>
<td>Present Speech</td>
<td>Present progressive</td>
<td>am/is/are {PRP.}</td>
<td>She is walking.</td>
</tr>
<tr>
<td></td>
<td>Past Speech</td>
<td>Future progressive</td>
<td>will be {PRP.}</td>
<td>She will be walking.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>to be {PRP.}</td>
<td>to be {PRP.}</td>
<td>To be walking.</td>
</tr>
<tr>
<td>During-</td>
<td>Future Speech</td>
<td>Past perfect progressive</td>
<td>had been {PRP.}</td>
<td>She had been walking.</td>
</tr>
<tr>
<td>After</td>
<td>Present Speech</td>
<td>Present perfect progressive</td>
<td>has/have been {PRP.}</td>
<td>She has been walking.</td>
</tr>
<tr>
<td></td>
<td>Past Speech</td>
<td>Future perfect progressive</td>
<td>will have been {PRP.}</td>
<td>She will have been walking.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>to have been {PRP.}</td>
<td>to have been {PRP.}</td>
<td>To have been walking.</td>
</tr>
<tr>
<td>Begin</td>
<td>Future Speech</td>
<td>“Began”</td>
<td>began {INFINITIVE}</td>
<td>She began to walk.</td>
</tr>
<tr>
<td></td>
<td>Present Speech</td>
<td>“Begins”</td>
<td>begins {INFINITIVE}</td>
<td>She begins to walk.</td>
</tr>
<tr>
<td></td>
<td>Past Speech</td>
<td>“Will begin”</td>
<td>will begin {INFINITIVE}</td>
<td>She will begin to walk.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>to begin {PRP.}</td>
<td>to begin {PRP.}</td>
<td>To begin walking.</td>
</tr>
<tr>
<td>Contains</td>
<td>Future Speech</td>
<td>Simple past</td>
<td>(SIMPLE PAST)</td>
<td>She walked.</td>
</tr>
<tr>
<td></td>
<td>Present Speech</td>
<td>Simple present</td>
<td>(SIMPLE PRESENT)</td>
<td>She walks.</td>
</tr>
<tr>
<td></td>
<td>Past speech</td>
<td>Simple future</td>
<td>will {INFINITIVE}</td>
<td>She will walk.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>(INFINITIVE)</td>
<td>(INFINITIVE)</td>
<td>To walk.</td>
</tr>
<tr>
<td>Before</td>
<td>Future Speech</td>
<td>“Posterior”</td>
<td>was/were going {INF}</td>
<td>She was going to walk.</td>
</tr>
<tr>
<td></td>
<td>Present Speech</td>
<td>am/is/are going {INF}</td>
<td>She is going to walk.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Past Speech</td>
<td>Future</td>
<td>will be going {INF}</td>
<td>She will be going to walk.</td>
</tr>
<tr>
<td></td>
<td>Modal Infinitive</td>
<td>to be going {INFINITIVE}</td>
<td>To be going to walk.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Tense/aspect assignment and realizer constructions for describing an action event from a particular perspective and speech time. “PR.P.” means “present participle.”
for walk\textsc{(Agent, Destination)}, which we derive from the VerbNet frame for “walk,” the result is “Mary begins to walk to the store;” similarly, (4.2) is realized as Julia is hungry via the During perspective. Narration mode invokes past-tense verbs.

### 4.4.5 Expressing single events from a reference interval

Just as events occur over intervals, rather than single points, so too can reference times. One may need to express what occurred when “Julia entered the room” (a non-instantaneous action) or “yesterday evening.” Our system allows annotators to view intervals in snapshot mode to get a sense of what happens over a certain time span.

The semantics of reference intervals have been studied as extensions to Reichenbach’s point approach. Dowty [1979, p.152], for example, posits that the progressive fits only if the reference interval is completely contained within the event interval. Following this, we construct an alternate lookup table (Table 4.9) for assigning the perspective of an event from a reference interval. Table 4.8 then applies in the same manner. In snapshot mode, the speech time $S$ also occurs over an interval (namely, $R$), and Present Speech is still used. In narration mode, $S$ is assumed to be a point following all event and reference intervals. In our running example, narrating the interval (1,7) results in “Mary walked to the store” and “Julia began to be hungry,” using the Contains and Begin perspectives respectively.

The notion of an unbounded reference interval, while unusual, corresponds to a typical perspective if the event is either bounded or unbounded in the opposite direction. These scenarios are illustrated in Table 4.9. Less intuitive are the cases where event and reference

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Relations</th>
<th>Perspective</th>
<th>Diagram</th>
<th>Relations</th>
<th>Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1 \geq E_2$</td>
<td>$R_1 \geq E_2$</td>
<td>After</td>
<td>$E_1 \leq E_2$</td>
<td>$E_1 &lt; R_1$</td>
<td>During</td>
</tr>
<tr>
<td>$R_2 \geq E_2$</td>
<td>$E_1 \geq R_2$</td>
<td>Contains</td>
<td>$E_1 \geq E_2$</td>
<td>$E_1 \geq R_2$</td>
<td>Before</td>
</tr>
</tbody>
</table>

Table 4.9: Perspective assignment for describing an event from an assigned perspective.
intervals are unbounded in the same direction. Perspective assignments for these instances are described in Table 4.10 and emphasize the bounded end of $R$. These situations occur rarely in this generation context.

**Event Subintervals**

We do not always want to refer to events in their entirety. We may instead wish to refer to the beginning, middle or end of an event, no matter when it occurs with respect to the reference time. This invokes a second reference point in the same interval [Comrie, 1985, 128], delimiting a subinterval. Consider “John searches for his glasses” versus “John continues to search for his glasses”—both indicate an ongoing process, but the latter implies a subinterval during which time, we are expected to know, John was already searching.

Our handling of subintervals falls along four alternatives that depend on the interval $E_1..E_2$, the reference time $R$ and the subinterval $E'_1..E'_2$ of $E$, where $E'_1 \geq E_1$ and $E'_2 \leq E_2$.

1. **During-After.** If $E'$ is not a final subinterval of $E$ ($E'_2 < E_2$), and $R = E'_2$ or $R$ is a subinterval of $E$ that is met by $E'$ ($R_1 = E'_2$), the perspective of $E'$ is defined as **During-After.** In Table 4.8, this invokes the perfect-progressive tense. For example, viewing example (4.1) with $E' = (2, 4)$ from $R = 4$ in narration mode (Future Speech) would yield “Mary had been walking to the store.”

2. **Start.** Otherwise, if $E'$ is an initial subinterval of $E$ ($E'_1 = E_1$ and $E'_2 < E_2$), the perspective is defined as **Start.** Not shown in Table 4.8, the construction for this case
reassigns the perspective to that between \( R \) and \( E' \). Our realizer reassigns the verb predicate to begin (or become for statives) with a plan to render its only argument, the original proposition, in the infinitive tense. For example, narrating (4.2) with \( E' = (1, 2) \) from \( R = 3 \) would invoke the After perspective between \( R \) and \( E' \), yielding “Julia had become hungry.”

3. **Continue.** Otherwise, and similarly, if \( E \) strictly contains \( E' \) (\( E_1' > E_1 \) and \( E_2' < E_2 \)), we assign the perspective **Continue.** To realize this, we reassign the perspective to that between \( R \) and \( E' \), and reassign the verb predicate to “continue” (or “was still” for statives) with a plan to render its only argument, the original proposition, in the infinitive: “Mary had continued to walk to the store” for (4.1) with \( E' = (3, 4) \) and \( R = 7 \).

4. **End.** Otherwise, if \( E' \) is a final subinterval of \( E \) (\( E_1' > E_1 \) and \( E_2' = E_2 \)), we assign the perspective **End.** To realize this, we reassign the perspective to that between \( R \) and \( E' \), and reassign the verb predicate to “stop” (or “finish” for cumulative achievements). Similarly, the predicate’s argument is the original proposition rendered in the infinitive.

### 4.4.6 Expressing multiple events in alternate timelines

This section covers more complex situations involving alternate timelines—the feature of our representation by which an event in the main timeline can refer to a second frame of time. Other models of time have supported similar encapsulations [Crouch and Pulman, 1993; Mani and Pustejovsky, 2004]. The alternate timeline can contain references to actual events or modal events (imagined, obligated, desired, planned, etc.) at earlier or later time states with respect to its **attachment point** (Section 4.3.4) on the main timeline. This is primarily used in practice for modeling dialogue acts, but it can also be used to place real events at uncertain time states in the past (e.g., the present perfect is used in a reference story being encoded).

**Reassigning Temporal Focus**

Ogihara [1995] describes dialogue acts involving changes in temporal focus as “double-access sentences.” We now consider a method for planning such sentences in such a way that the refocusing of time (the reassignment of \( R \) into a new context) is clear, even if it means changing tense and aspect mid-sentence. Suppose Mary were to declare that she would buy some eggs because of Julia’s hunger, but before she returned from the store, Julia filled up on snacks. If this speech act is described by a character later in the story, then we need to carefully separate what is known to Mary at the time of her speech from what is later known at \( R \) by the teller of the episode. Mary sees her purchase of eggs as a possible future, even though it may have already happened by the point of retelling, and Mary does not know that Julia’s hunger is to end before long.

Following Hornstein’s treatment of these scenarios [Hornstein, 1990], we attach \( R' \), the reference time for Mary’s statement (in an alternate timeline), to \( E_{speech} \), the event of her...
speaking (in the main timeline). The act of buying eggs is a hypothetical event $E'_{buy}$ that falls after $R'$ on the alternate (modal) timeline. $S$ is not reassigned.

Figure 4.20 shows both timelines for this example. The main timeline is shown on top; Mary’s speech act is below. The attachment point on the main timeline is, in this case, the speech event $E_{speech}$; the attachment point on an alternate timeline is always $R'$. The placement of $R$, the main reference point, is not affected by the alternate timeline. Real events, such as Julia’s hunger, can be invoked in the alternate timeline (e arcs, Section 3.3.1, as drawn with a vertical line from $E_{hunger}$ to an $E'_{hunger}$ without an $E'_{2}$ known to Mary) but they must preserve their order from the main timeline.

The tense assignment for the event intervals in the alternate timeline then proceeds as normal, with $R'$ substituting for $R$. The hypothetical “buy” event is seen in Before perspective, but past tense (Future Speech), giving the posterior (future-of-a-past) tense. Julia’s hunger is seen as During as per Table 4.7. Further, we assert that connectives such as “because” do not alter $R$ (or in this situation, $R'$), and that the $E'_{buy}$ is connected to $E'_{hunger}$ with a causality edge.

The result is: “Mary had said that she was going to buy eggs because Julia was hungry.” The subordinate clause following that sees $E_{buy}$ in the future, and $E'_{hunger}$ as ongoing rather than in the past. It is appropriately ambiguous in both the symbolic and rendered forms whether $E'_{buy}$ occurs at all, and if so, whether it occurs before, during or after $R$. A discourse planner would have the responsibility of pointing out Mary’s mistaken assumption about the duration of Julia’s hunger.

We assign tense and aspect for quoted speech differently than for unquoted speech. Instead of holding $S$ fixed, $S'$ is assigned to $R'$ at the attachment point of the alternate timeline (the “present time” for the speech act). If future hypothetical events are present, they invoke the Past Speech constructions in Table 4.8 that have not been used by either narration or snapshot mode. The content of the quoted speech then operates totally independently of the speech action, since both $R'$ and $S'$ are detached: “Mary said/says/was saying, ‘I am going to buy eggs because Julia is hungry.’”

The focus of the sentence can be subsequently reassigned to deeper nested timelines as necessary (attaching $E'$ to $R''$, and so on). Although the above example uses subordinate
clauses, we can use this nesting technique to construct composite tenses such as those enumerated by Halliday [1976]. To this end, we conjugate the Modal Infinitive construction in Table 4.8 for each alternate timeline. For instance, Halliday’s complex form “present in past in future in past in future” (as in “will have been going to have been taking”) can be generated with four timelines in a chain that invoke, in order and with Past Speech, the After, Before, After and During perspectives. There are four Rs, all but the main one attached to a previous E (Figure 4.21).

Subjunctives and Conditionals

We finally consider tense and aspect in the case of subjunctive and conditional statements (if-thens), which can appear in alternate timelines (Section 4.3.4). The relationship between an if clause and a then clause is not the same as the relationship between two clauses joined by because or when. The then clause—or set of clauses—is predicated on the truth of the if clause. As linguists have noted [Hornstein, 1990, p.74], the if clause serves as an adverbial modifier, which has the effect of moving forward the reference point to the last of the if event intervals (provided that the if refers to a hypothetical future). Consider the sentence: “If John were to fly to Tokyo, he would have booked a hotel.” A correct model would place \( E'_\text{book} \) before \( E'_\text{fly} \) on an alternate timeline, with \( E'_\text{fly} \) as the if. Since \( \text{were to fly} \) is a hypothetical future, \( R' < E'_\text{fly} \). During regeneration, we set \( R' \) to \( E'_\text{fly} \) after rendering “If John were to fly to Tokyo,” because we begin to assume that this event transpired. If \( R' \) is left unchanged, it may be erroneously left before \( E'_\text{book} \): “Then he would be going to book a hotel.”

Our annotation interface allows users to mark one or more events in an alternate timeline as if events. If at least one event is marked, all if events are rendered in the subjunctive mood, and the remainder are rendered in the conditional. For the if clauses that follow \( R' \), \( S' \) and \( R' \) itself are reassigned to the interval for each clause in turn. \( R' \) and \( S' \) then remain at the latest if interval (if it is after the original \( R' \)) for purposes of rendering the then clauses. In our surface realizer, auxiliary words \textit{were} and \textit{would} are combined with the Modal Infinitive constructions in Table 4.8 for events during or following the original
As an example, consider an alternate timeline with two statives whose start and end points are the same: “Julia is hungry” and “Julia is unhappy.” The former is marked if. Semantically, we are saying that \( \text{hungry}(Julia) \rightarrow \text{unhappy}(Julia) \). If \( R' \) were within these intervals, the rendering would be: “If Julia is hungry, then she is unhappy” (Contains/ Present Speech for both clauses). If \( R' \) were prior to these intervals, the rendering would be: “If Julia were to be hungry, then she would be unhappy.” This reassigns \( R' \) to \( E_{\text{hungry}} \), using were as a futurative and would to indicate a conditional. Because \( R' \) and \( S' \) are set to \( E_{\text{hungry}} \), the perspective on both clauses remains Contains/ Present Speech. Finally, if both intervals are before \( R' \), describing Julia’s previous emotional states, we avoid shifting \( R' \) and \( S' \) backward: “If Julia had been hungry, then she had been unhappy” (After perspective, Future Speech for both statives).

The algorithm is the same for event intervals. Take (4.1) and a prior event where Mary runs out of eggs:

\[
\text{runOut}(Mary, \text{eggs}, 0, 1) \quad (4.3)
\]

Suppose they are in an alternate timeline with attachment point \( 0' \) and (4.1) marked if. We begin by realizing Mary’s walk as an if clause: “If Mary were to walk to the store.” We reassign \( R' \) to \( E_{\text{walk}}, (2,6) \), which diverts the perception of (4.3) from Begins to After: “She would have run out of eggs.” Conversely, suppose the conditional relationship were reversed, with (4.3) as the only if action. If the attachment point is \( 3' \), we realize (4.3) first in the After perspective, as \( R' \) does not shift backward: “If Mary had run out of eggs.” The remainder is rendered from the During perspective: “She would be walking to the store.” Note that in casual conversation, we might expect a speaker at \( R = 3 \) to use the past simple: “If Mary ran out of eggs, she would be walking to the store.” In this case, the speaker is attaching the alternate timeline at a reference interval that subsumes (4.3), invoking the Contains perspective by casting a net around the past. We ask our annotators to select the best attachment point pursuant to their understanding of the story content.

### 4.4.7 Discussion

As we mentioned earlier, we are describing two separate methods with a modular relationship to one another. The first is an abstract mapping from a conceptual representation of time in a narrative, including interval and modal logic, to a set of 11 perspectives, including the 7 listed in Table 4.8 and the 4 introduced in Section 4.4.5. These 11 are crossed with three scenarios for speech time to give a total of 33 tense/aspect permutations. We also use an infinitive form for each perspective. One may take these results and map them from other time representations with similar specifications.

The second result is a set of syntactic constructions for realizing these permutations in our text generation module. Our focus here, as we have noted, is not fluency, but a surface-level rendering that reflects the relationships (and, at times, the ambiguities) present in the encoding. We consider variations in modality, such as an indicative reading as opposed to a conditional or subjunctive reading, to be at the level of the realizer and not separate classes of tenses.
It has always been the goal in surface realization to generate sentences from a purely semantic representation. Our approach to the generation of tense and aspect explores a rule-based approach based on temporal intervals. We have applied prior work in linguistics and interval theory and tested our approach in an interactive narrative encoding tool. Our method handles reference intervals and event intervals, bounded and unbounded, and extends into subintervals, modal events, conditionals, and direct and indirect speech where the temporal focus shifts.

4.5 Conclusion

In this chapter, we described the development of Scheherazade as a software system capable of performing inference over general semantic networks, implementing the particular semantics of the SIG model of discourse relations we introduced in Chapter 3, and eliciting semantic encodings of narratives from trained annotators through a graphical user interface. We gave particular emphasis to the process of propositional modeling, including our use of external linguistic resources to populate a database of linguistic knowledge. We also described a textual generation module that is capable of generating feedback text from a SIG encoding, in whole or in part; in the context of this work, we described a model for the assignment of tense and aspect in narrative discourse that covers the linguistic scenarios made possible by the interval-based representation of time and modality employed by the SIG. To our knowledge, this combination of a formal representation of narrative time and mode and a generative model of tense and aspect is a novel contribution. We believe that Scheherazade can act as a foundational platform for future work in story representation and analysis, as an API allows third-party tools to utilize the library for storing, reading and comparing encodings.

In Chapter 5, we evaluate the Scheherazade implementation in the context of a series of corpus collection projects, and describe algorithms that allow us to leverage the representation to find similarities and analogies between encoded narratives.
Chapter 5

Collections and Experiments

Our premise, which we laid out in Chapter 1, is that a plethora of online discourse has a narrative component. An algorithm capable of finding thematic similarities between stories can greatly assist us in our need to filter, search, and otherwise organize the many stories to which we are exposed on a daily basis, from news articles to fiction and personal communication. Chapters 2 and 3 explored several types of "narrative components," with the former specializing in conversational networks found in literary fiction, and the latter proposing a broader set of relations describing temporal, causal, and agent-oriented relationships that provide for narrative cohesion.

The latter approach, which we call the Story Intention Graph, aims to be a middle ground between surface text and a formal model of plot, action and character. This final chapter explores both ends of its reach: On one end, we describe a corpus collection which brings SIG annotation to trained annotators (using the Scheherazade tool); on the other end, we explore some of the thematic insights that we can draw from applying formal inference rules to SIG encodings. The task that we apply to the SIG is that of identifying similarities and analogies between stories, a process that is crucial for all of the applications we have mentioned. If we can determine whether two stories are similar or analogous (isomorphic in structure), we can more easily interpret new stories by their multifaceted relationships with known stories. Much like a trained language model allows us to recognize n-grams as being more than the sum of their parts, a data bank of encoded stories would let us identify "narrative idioms" that recur and are likely to appear in future stories. As language models have been widely influential in tasks such as information retrieval, parsing and speech recognition at the clause level, a model of narrative idioms, and a method for accurately identifying them in previously unseen text and speech, would be influential at the discourse level. This discourse-clause distinction also distinguishes our task from that of measuring sentence-level similarity, which has been investigated by projects such as Columbia SimFinder [Hatzivasiloglou et al., 2001].

In terms of process, we aim to develop procedural methods of identifying pairwise similarities between SIG encodings that relate to events (propositional content), goals, plans and affectual impact. We also strive to find overall trends that hold across a corpus of encodings. In a sense, we wish to accomplish automatically the type of structuralist analysis of similarities and trends that Propp performed on Russian folk-tales [Propp, 1969] and