University of Alberta

DIALOGUE PATTERNS IN COMPUTER ROLE-PLAYING GAMES

by

Jeffrey D Siegel 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2007

Your file Votre référence
ISBN: 978-0-494-30024-4
Our file Notre référence
ISBN: 978-0-494-30024-4

Abstract

Today's computer role-playing games (CRPGs) have ever increasing sophisticated and complex elements, including rich and dynamic character conversations. CRPGs such as Neverwinter Nights use manual scripting to control the flow of these conversations. These scripts can be confusing and time consuming to game designers with no programming experience. This dissertation presents a new dialogue pattern model to construct conversations in the Neverwinter Nights CRPG. This model uses a more compact and concise representation than the model used by the Neverwinter Nights Aurora conversation editor. The scripts used to create dynamic conversations in the Aurora conversation model are replaced with generative design patterns. These design patterns generate the scripting code automatically, preventing the game designer from making any scripting mistakes. A case study analyzes the effectiveness of both models by using five metrics which compare the models against several criteria. The dialogue pattern model is shown to be easier to use.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

1.1 Computer Role-playing Games

Role-playing games are a popular and complex type of computer game. Computer role-playing games (CRPGs) involves a grand, intricate story, much like a novel. The player controls a player character (PC), or a group of characters, and using the PC explore a world that involves battling creatures, solving puzzles, completing quests or objectives, and speaking to computer controlled non-player characters (NPCs). Unlike a novel, the player can make choices during the game, and each choice can affect the outcome of the story. The story is revealed in small sections called quests. Each quest gives the player a specific task to accomplish, and the player may be involved in multiple quests simultaneously. If the game world is large enough, many quests will have no bearing on the main story line, but serve to add depth to the world and entertain the player.

With new, powerful gaming consoles, CRPGs can be played on either personal computers or gaming consoles. On a personal computer, the player controls the PC's movement and actions by clicking with the mouse or pressing arrow keys on the keyboard. On consoles, the player uses the console controller to direct the character's actions. The PC can interact with objects in the game, including props such as tables and chairs. For example, clicking on a lever with the mouse may cause a door to open. If the player clicks on a friendly NPC, it is possible to initiate a conversation with that NPC and a dialogue will appear, giving the player the choice of what they can say.

Combat is a large aspect of CRPGs. During an adventure, the PC may encounter hostile creatures. Rules exist in the game to determine who can attack first, if a combatant gets hit, and how much damage they receive – usually represented as a number. Many role-playing games are fantasy-based, so they include magic and spell-casting systems as well.

Figure 1.1 shows a conversation scene in Knights of the Old Republic, Bioware's futuristic RPG based in the Star Wars universe [12]. Here the PC is conversing with an NPC in the game. The player has a list of responses that the PC can say to the NPC. Figure 1.2 shows a combat scene in Oblivion, Bethesda's latest game in the Elder Scrolls Saga [19]. The player is in a first person view fighting a hostile NPC. The player can attack with an equipped weapon by clicking the mouse.

Figure 1.1: Conversation with an NPC in Knights of the Old Republic

Figure 1.2: Locked in combat with a hostile NPC in Oblivion

CRPGs use abstraction to represent properties of the world. Characters have integers to represent ability stats, such as strength, intelligence, or charisma. They have skills that determine how well they can perform specific tasks. For example, during conversations the PC's persuasion skill determines the PC's effectiveness in convincing the NPC to provide extra information or rewards. Fighting in battles is abstracted by using either a turn-based or real-time combat system. Formulas calculate hits and misses, item bonuses, and any damage received during combat. In most games, if a character's hit points (health) drop to 0, the character is considered dead.

The majority of the time in the game is spent performing quests. Most quests are assigned, and later completed, by conversing with NPC characters. Conversations are an important part of role-playing games. They not only add life to the NPCs, but also impart important information to allow the player to progress in the game by completing quests and receive directions for new areas to explore. Unlike traditional stories, interactive stories like CRPGs have no narrative text to advance the story. Instead, CRPGs rely on conversations and journal entries to give narrative to the players.

The nature of the PC's conversation with an NPC can change over time as the game state changes. Conversely, conversations with NPCs can change the game state. For example, an NPC greets the PC on the first conversation, but if the PC is insulting, the NPC will refuse any further attempts to converse. If the PC is looking for treasure and does not know the location, the NPC can give the treasure's location in a conversation. If the PC already knows the treasure's location, the NPC won't say anything about the location. Similarly, the PC can choose a quest from the NPC, which changes the game state by opening a new area to explore.

State-of-the-art CRPGs are becoming more sophisticated. The worlds are larger, with more creatures and objects. AI systems are growing more complex, with an increasing expectation for characters that have a rich set of behaviours that project a feeling of intelligence. Intelligent characters require intelligent conversations, and intelligent conversations require an effective construction tool. This dissertation describes a new method of constructing conversations by using generative design patterns.

1.2 Interactive Story-telling

CRPGs are a form of interactive story. Like written stories, interactive stories have a plot with settings, protagonists and antagonists. However, interactive stories differ by allowing the player or participant to affect the outcome of the story through their actions. It can be as simple as choosing a page number in the Choose Your Own Adventure books [5] or as complex as helping shape the story as a player in Pen-and-Paper Role-playing Games [6, 8]. Improv comedy, such as the hit show *Who's Line is it Anyways*, also has the story shaped by the participants. Several types of interactive storytelling involve a game master (or GM). The game master mediates the storytelling, and can be responsible for narrative flow, rules, engagement, environment, and the virtual world [21]. In CRPGs, the game master is normally replaced by a game engine and pre-generated content.

CRPGs are criticized as being finite in size and limiting in choices. The story is pre-defined, with the player only having a few possible story branches to choose from. The player still has the option to do extra side quests, but these normally serve to increase the character's power or provide an entertaining diversion rather than affecting the main story. For example, in Bioware's Jade Empire[9], the player can choose to be either good or evil. However once they make that choice, they follow a pre-determined story until the game ends. Similarly, conversations with NPCs are limited to a set number of statements and responses. The conversation is designed and built by authors before the game is played, and the player only has a few options when replying to what an NPC says.

Some work has been done to give players more options when interacting with NPCs. Microsoft Research has looked into applying NLP techniques to NPC dialogue [10, 11]. Each NPC would have a knowledge base that would change as the game state changes. When a PC initiates a conversation, the NPC uses the knowledge base to dynamically generate a conversation, complete with grammatically correct statements and a list of responses the PC can choose from.

The Façade project[14] goes one step further by creating a game that provides interactive drama. Instead of giving the PC only a list of pre-determined responses, the PC can now use the keyboard to enter any free-form English statement or question to communicate with the NPCs. The NPCs have sophisticated motivational-based artificial intelligence to determine how to respond to the PC and what actions to perform in the virtual world.

1.3 Neverwinter Nights

Neverwinter Nights is a role-playing game developed by Bioware Corp [1]. It has won numerous awards, including many game-of-the-year awards [17]. Based in the Forgotten Realms setting, it uses a modified version of the pen-and-paper Dungeons and Dragons D20 system for game rules and mechanics [4]. There are two parts to the game: the engine and modules. The game engine is responsible for rendering game objects and special effects, moving game objects, playing music and sound, and dispatching events to scripts, which in turn are executed by a virtual machine. Modules are files that contain story content, including map data, story objects, scripts, and conversation files. To play Neverwinter Nights, the player starts the game and selects a module (i.e. story). The player then selects the PC that will play through the module. The game engine then loads the module's scripts and game objects into memory and the game begins. Figure 1.3 shows a screenshot of a Neverwinter Nights module.

Neverwinter Nights includes an official campaign story comprised of 7 modules. The game's two expansions, Shadows of Undrentide and Horde of the Underdark, provide a further six 6 modules. In addition to the content contained in the 13 official modules, Neverwinter Nights also includes tools to allow players and designers to create their own content. Consequently, a large community has formed to share ideas, provide help with scripting, author new modules, and play modules

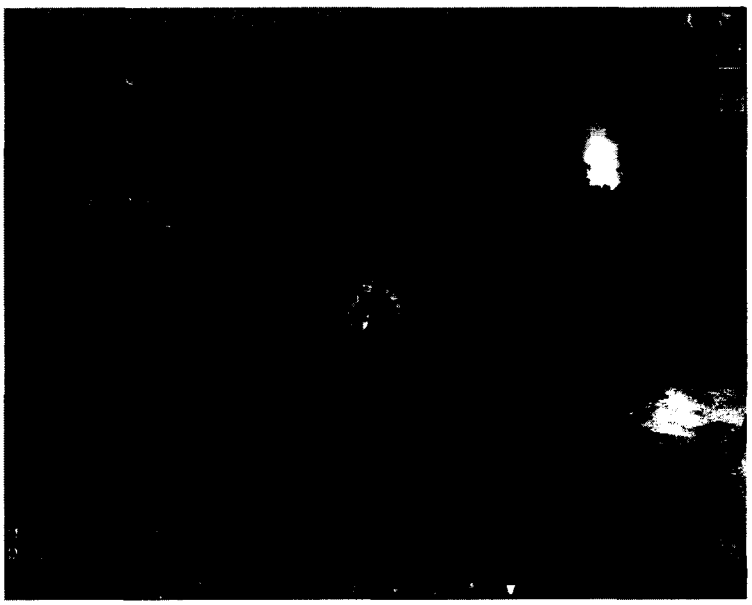


Figure 1.3: The PC in a city setting.

created by other players and amateur designers. Bioware's community website is the nexus of this community[18].

In Figure 1.3, the PC, Captain Adventure, is in a large city complete with fountains, pillars, and a fire from unruly residents. The interface includes buttons on the bottom that allow the player to make the PC perform actions such as casting a spell or quaffing a potion. The top right corner holds the PC's portrait, life bar (percentage of hit points), and buttons to access windows that provide information such as the PC's statistics, learned spells, and journal entries for quests. In front of the PC are NPCs that wander the city. The player can initiate conversations with these NPCs by clicking on them.

Figure 1.4 shows a PC in conversation with an NPC named Ras Whisperwind. The PC has a chance to persuade Ras to pay extra gold for an escort to the nearest village. If the PC is successful, Ras will hand over the additional gold and begin following the PC.

1.4 Aurora Toolset

One of Neverwinter Night's unique features is the Aurora toolset included with the game. The toolset allows the author to edit module resources, including 1) area maps, 2) game objects, 3) conversations, and 4) scripts. A conversation file is an example of a resource in a module. Conversations are tree structures and can be edited by a conversation editor in the toolset. Each statement made by a PC or NPC is represented by a conversation node in the tree. Figure 1.5 shows an example conversation tree in the Aurora toolset conversation editor.

A conversation file controls how a conversation flows. Conversation nodes that represent NPC

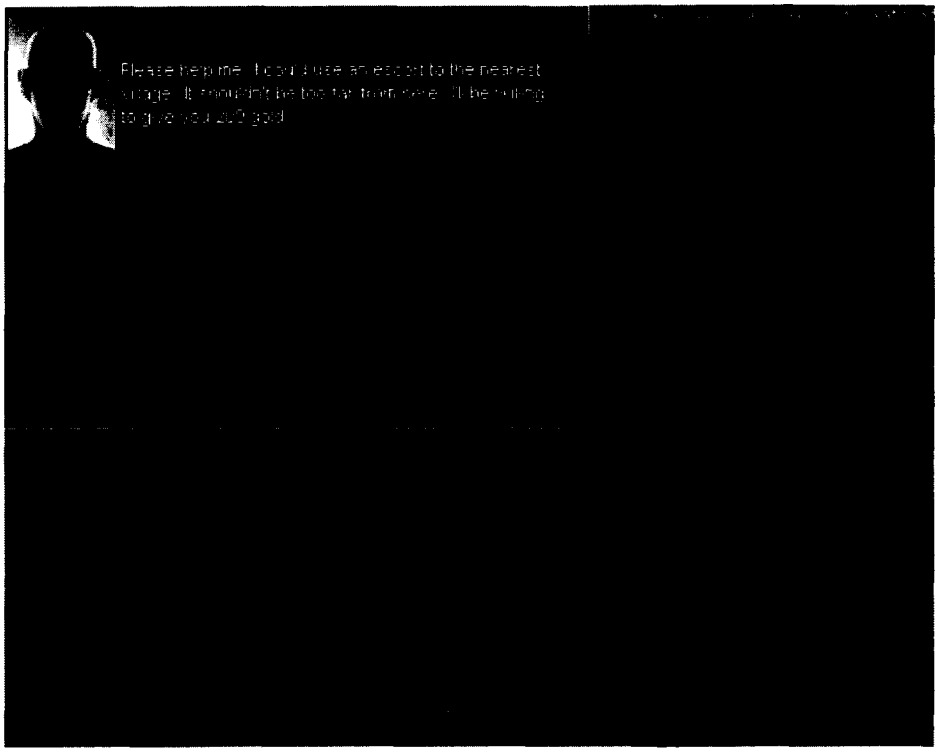


Figure 1.4: The PC conversing with a lost merchant.

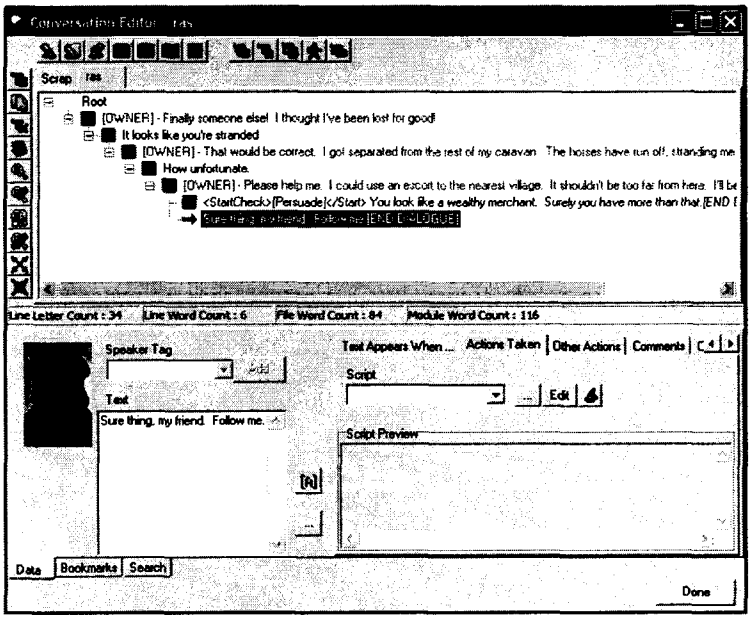


Figure 1.5: Conversation file for the NPC Ras Whisperwind.

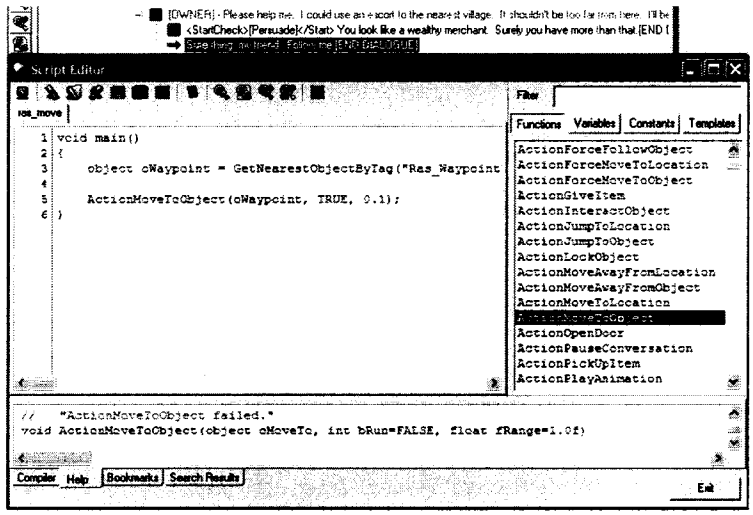


Figure 1.6: A script that causes an NPC perform actions.

statements are followed by children nodes that represent PC responses to the NPC statement. Like many of the game objects, scripts can be attached to nodes in the conversation file. During a conversation, these scripts can be executed, allowing the game state to change. For example, Figure 1.6 shows a typical script in the Aurora toolset's script editor. The script is attached to the selected conversation node in Figure 1.5 at the end of the conversation tree. The script moves the NPC to a waypoint at another location on the map.

For maps and game objects, Aurora provides a CAD-like interface where the author can paint terrain and objects using the mouse. A module is composed of several areas, each containing a map. Maps are divided into a grid of tiles. The author picks a terrain tileset and can paint these terrain tiles onto the map. Figure 1.7 shows a wooden building being painted into an area with a forest tileset. After building all the terrain, the author can then choose from a wide selection of game objects to populate the newly constructed map.

1.4.1 Toolset Interface and Area Maps

The interface is divided into three main sections, shown in Figure 1.8. The left-most section, labeled **1**, displays the area information. A module is divided into several areas, each with a separate map. The areas are the first module resource. Each area can contain game objects, such as creatures and doors. The center section, labeled **2**, displays the current area as it would be rendered in the game. Below are camera controls to move the camera around the area. The author can click on visible objects in this window to move them, orient them, or access their properties. The right-most section, labeled **3**, displays the game object palette. Here the author can select game objects to paint them into the area. At the top are icons that show the different object categories. Below are two buttons, one to access the standard palette and the other the custom palette. The standard palette includes all

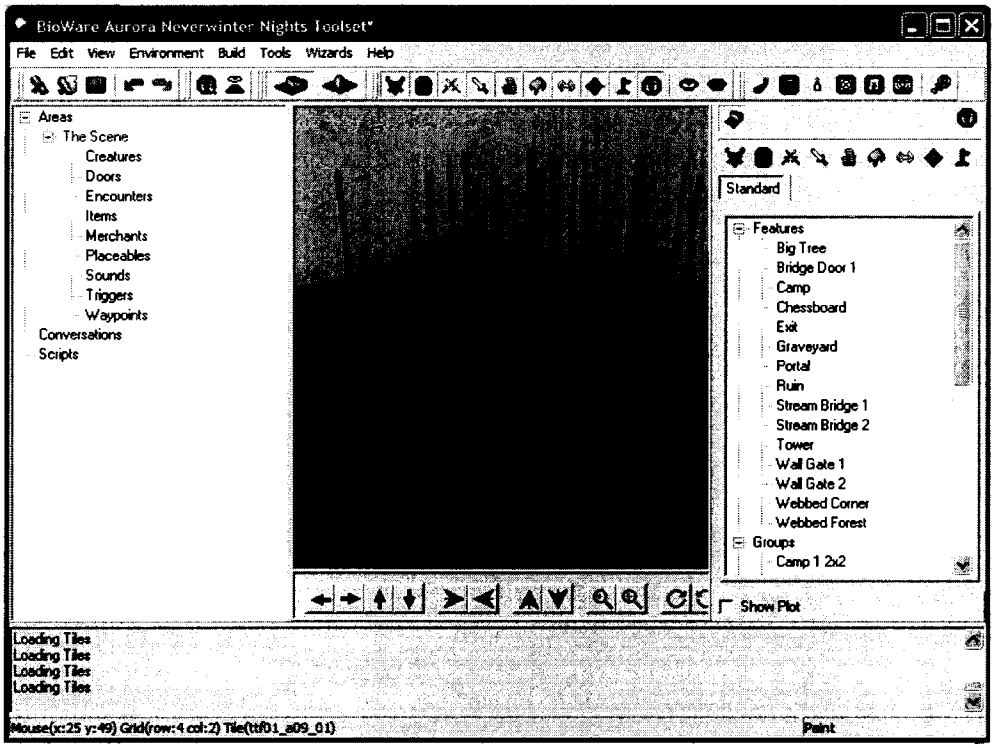


Figure 1.7: Painting a building into an area with the Aurora toolset.

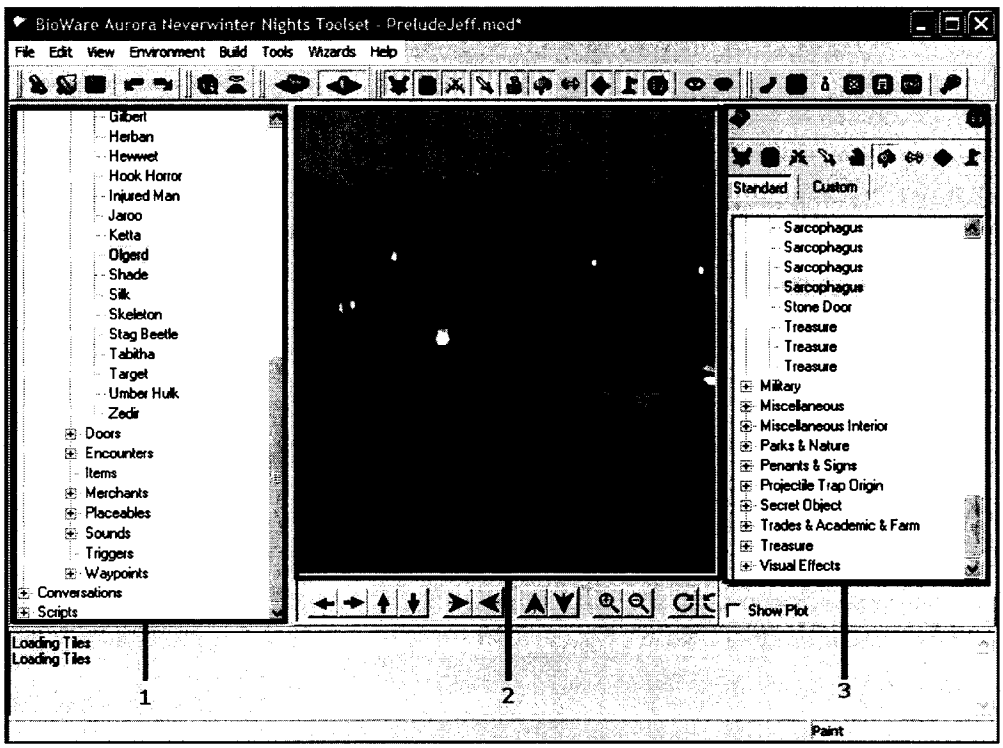


Figure 1.8: The three main sections of the Aurora toolset.

pre-built game objects that are provided by the game. The custom palette holds both newly created game objects and standard objects customized by the author.

Although most objects are visible while playing the game, such as characters and inanimate objects, there are also invisible objects. The module, areas in the module, and waypoints are all examples of invisible objects. In the case of waypoints, these are visible in Aurora, but invisible when the game is played. For example, Figure 1.8 shows a waypoint object represented as a flag with an arrow. Waypoints represent an orientation and a location on a map. Waypoints are used as reference points, such as teleporting a PC to a new location.

1.4.2 Game Objects

The second major module resource contains game objects Neverwinter Nights supports the following object categories. The first 6 categories are scriptable, and the last 3 are not.

- **Creatures** - Creatures represent sentient entities. They can move, participate in combat, converse, and can hold item objects in their inventory. NPCs, the PC, and monsters are all creatures.
- **Doors** - Doors can be painted in certain tiles that have a doorway. Doors prevent creatures from passing through, and can be opened, closed, locked, and bashed down.
- **Placeables** - Placeables are inanimate objects such as chairs, tables or chests. Most are used for decoration, but some are containers (e.g. chests) that can hold item objects. The author can customize a placeable to be interactive, such as clicking on a statue to access its inventory.
- **Triggers** - Triggers are invisible polygons painted on the map. Events are fired when a creature enters or exits the trigger, allowing the author to change the game state. Examples include defining a region to have an effect such a teleporting the intruder, spawning a creature, or springing a trap.
- **Encounters** - Encounters are special triggers that spawn creatures when an intruder enters the trigger. An encounter automatically scales the level and quantity of spawned creatures, based on the level and power of the PC. Encounters are used extensively in the Neverwinter official campaign to create monsters for the PC to fight during the progression through the game.
- **Merchants** - Merchants are invisible objects that represent a store where the PC can purchase items and equipment. A merchant object can contain item objects much like a container. The author then connects a merchant object with an NPC character that acts as the merchant. The PC can then buy from the store by initiating a conversation with the NPC character.
- **Sounds** - Sound objects are invisible with the sole purpose of broadcasting a sound in a particular part of the module. Each sound object has a broadcasting radius and the sound effect tapers that off as the PC moves farther away.

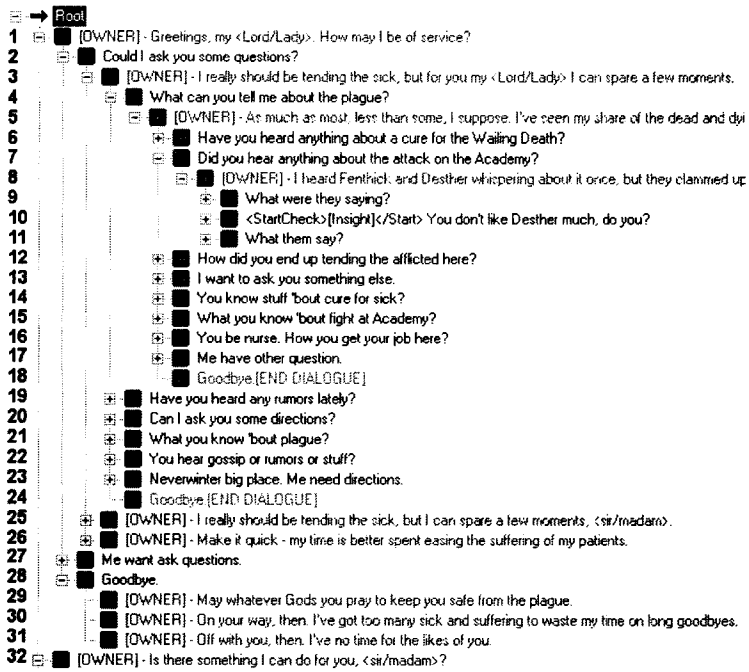


Figure 1.9: A conversation tree for a nurse NPC in Chapter 1 of Neverwinter Nights.

- **Items** - Items are non-scriptable objects that exist in creature or placeable inventories, or found on the ground. Items include weapons, armour, potions, and books. Many items can be equipped by a PC or NPC, giving a boost to their abilities. Items can be bought from and sold to merchants.

- **Waypoints** - Waypoints are invisible objects that the author can paint on the map. Waypoints are used as markers for controlling creature movement. For example, a guard patrolling a treasure chest follows a set of waypoints clustered around the treasure chest.

1.4.3 Conversations

Conversation files are the third major module resource. A conversation is viewed as a tree, with each branch representing a possible line of dialogue the player can see when playing the game. Figure 1.9 shows a conversation from an NPC in Chapter 1 of Neverwinter Night's official campaign story. A tree of nodes is visible, each on a separate line with the statement text. NPC remark nodes are coloured in red and PC choice nodes are coloured in blue. NPC nodes always contain PC child nodes, and similarly PC nodes contain NPC child nodes.

Semantically, NPC nodes differ from PC nodes in that only one can be displayed in a conversation at any one time. If a PC node contains several NPC nodes, then the game engine must select only one of the NPC nodes. To do this, the engine evaluates a **When** script on each NPC node, one at a time. A **When** script is evaluated to determine whether the node is to be displayed or hidden.

The first NPC node with a **When** script that returns *TRUE* is selected to be displayed. In Figure 1.9, there are three NPC children under the "Goodbye" PC node (line 28). If the PC selected this choice during the game, the game engine would evaluate the **When** script of the first NPC child. In this case, the script returns *TRUE* if the PC has a high charisma ability score, *FALSE* otherwise. If the script returns *TRUE*, the NPC remark on line 29 is displayed. If the script returns *FALSE*, the engine evaluates the **When** script for the next NPC child and so on. If all **When** scripts return *FALSE*, the last NPC child is selected by default.

In contrast, if an NPC node contains multiple PC children nodes, any number of children can be displayed as PC nodes simultaneously. These PC nodes are a list of choices where the PC can choose the appropriate response. A **When** script on a PC response node will hide the node from the list of choices if the script evaluates to *FALSE*. For example, in Figure 1.9 the PC node on line 10 has an attached **When** script, which is indicated by green "~" in the node's blue square icon. The script only returns *TRUE* if the PC has a high wisdom[1] ability score. For players without high wisdom, this response will be absent when they reach that point in the conversation.

In addition to **When** scripts, conversation nodes can also have **What** scripts. These scripts are executed after a conversation node is displayed in a conversation. In the case of a PC node, the script is executed when the PC selects that node as a response.

Figure 1.9 also shows nodes that are gray in colour on lines 18 and 24. These are link nodes. They convert the conversation from a tree to a graph by allowing branches of the conversation to lead into other branches. For example, the "Goodbye" link on lines 18 and 24 gives the PC the option to pick the goodbye choice found at line 28 at several points during the conversation. The author does not have to replicate the goodbye sub-tree in multiple locations of the tree. Links also allow the PC to ask the NPC to repeat questions or explanations without duplication a sub-tree of conversation nodes.

Conversations are constructed using a simple set of operations. A author creates the conversation by adding one conversation node per operation. After a node is created, the author can set the remark text and properties, as well as attach **When** and **What** scripts. Finally, the author can delete a node and its subtree. The complete list of operations follows:

- **Add Node** - Adds a new child node to the selected node.
- **Remove Node** - Deletes a node and all its descendants.
- **Edit Text** - Changes the node's statement text.
- **Edit Property** - Change a property of a node, including animation and journal entry updates.
- **Add Link** - Involves copying a node, then pasting it as a link node in some other location in the tree.

[1] Wisdom determines a character's intuition, insight, and overall knowledge.

- **Attach Script** - Opens a dialog box with a lists of scripts. The user selects one to attach as either a **When** script or **What** script.

1.4.4 Scripts

Scripts are the fourth major module resource. Each script is written in the Bioware-invented NWScript code which is a statically typed C-like language. The game engine exposes an API which the author can use in scripts to control the story. Scripts can also store integer, string, and object variables on any game object in the module. Figure 1.6 shows a script that is used in a conversation. This script forces the NPC to walk to a waypoint after talking with the PC.

Scripts are attached to game objects and conversation nodes. Objects respond to certain events that can be fired during the game. When an event is fired, the game engine looks up the corresponding script on the object responding to the event. The engine then executes the script. Once the script is completed, the engine continues running the game by processing new events.

The author can use the Aurora toolset to attach scripts to game objects. After selecting the desired object, the author picks an event that will execute the script. For example, the selected conversation node in Figure 1.5 responds to two events: **When** and **What**. In the figure, these events are represented by the "Text Appears When" and "Actions Taken" GUI tabs. The script can be attached by either typing the name of the script in a text field, or selecting the script from a list in a pop-up window. Figure 1.6 shows a **What** script that the author attaches to the conversation node. While conversation nodes only respond to two events, certain game objects respond to many events. A chest object, for example, responds to 13 events. Figure 1.10 shows the interface where the author can attach 13 possible scripts – one script for each event – to the chest object. Like attaching scripts to conversation nodes, a script can be attached to an object by typing the name of the script, or selecting it from a list by clicking the "..." button.

1.5 Aurora Toolset Deficiencies

While simple at first, the Aurora toolset becomes challenging to use when the author tries to create more sophisticated game object interactions. Any non-trivial player/object interaction requires the author to create a script by using the scripting editor [15]. Creatures have simple behaviours since interesting behaviours are not cost-effective with current tools [3]. Managing quests and the overall story requires the author to set and check a complicated set of esoterically-named variables spread across tens or even hundreds of scripts. Conversation trees can quickly become wide and deep, scattered with many link nodes. For the purpose of this dissertation, this section will focus on the shortcomings of the Aurora's conversation editor.

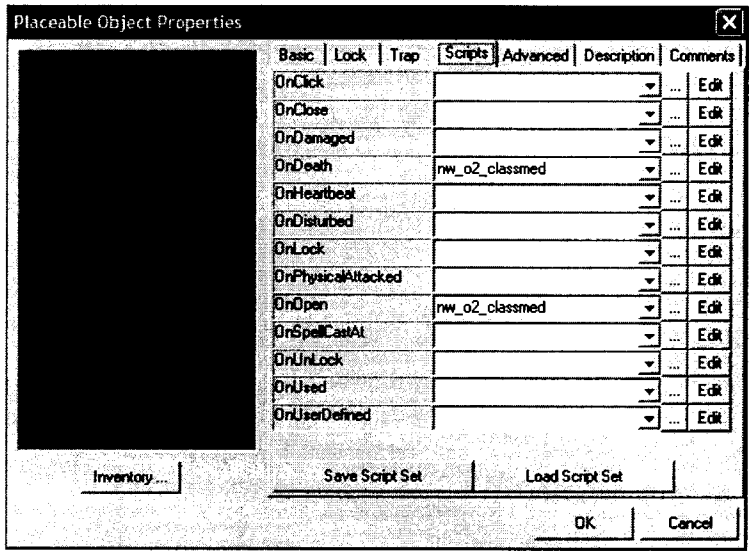


Figure 1.10: A chest object can have up to 13 scripts attached.

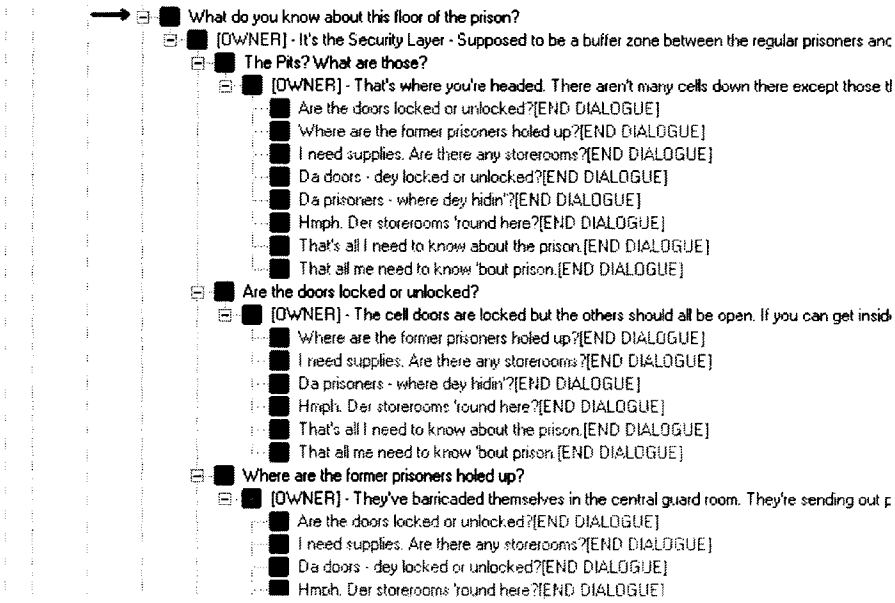


Figure 1.11: A conversation tree with many link nodes.

1.5.1 Conversation User Interface

The Neverwinter Nights official campaign story contains many large and complicated conversations. Figure 1.11 shows a portion of such a conversation for the Emernick character from Chapter 1 of the official campaign story. The complete conversation contains 176 nodes. This conversation has a depth of 10. The depth of the tree is the maximum number of levels needed to reach any leaf node from the root node. Each level of the conversation tree is indicated by a vertical line that connects sibling nodes. As the tree view is expanded to reveal lower levels, outer vertical lines become longer, and the sibling nodes move farther apart.

Once several levels of the tree are in view, it becomes difficult for the author to associate parent nodes with their children. This hampers the author from getting an overall view of the conversation. To get a better overview, the conversation needs to be abstracted. That is, hide unnecessary details of the conversation and only show the important details. The Aurora conversation editor only provides one mechanism for abstracting the conversation tree: expanding and collapsing conversation nodes. For example, in Figure 1.11, the conversation node on the first line marked by the arrow has no siblings NPC nodes visible. Instead, the screen is filled with the node's descendants, mostly link nodes. Since the link nodes are terminal, the author can get a more concise view of the tree by collapsing the link nodes into their parents. Unfortunately, collapsing a conversation node hides its entire sub-tree, including any important branches of conversation that the author may want to view. One of the contributions of this dissertation are better abstraction mechanisms presented in Chapters 3 and 4.

In the Aurora conversation editor, a author can create a link node that points to another part of the conversation tree. This allows the author to create repeatable sections of conversation, such as the PC asking a set of questions at several different points in the conversation. Many large conversations consist largely of links and it is common to see several groups of links, as shown in Emernick's conversation in Figure 1.11. Unfortunately, the Aurora conversation editor does not provide any visual indication of a link node's target node. The author must instead double-click the link to be redirected to the actual node. For large conversations, this may scroll the screen to the point where the original source link node is moved off the screen. With so many links in a single conversation, it is impossible for the author to get an overall view of the conversation. For example, the Emernick conversation (Figure 1.11), 51% of the entire conversation tree is composed of link nodes.

1.5.2 Conversation Scripting

As Section 1.4.3 describes, conversation nodes can have **When** and **What** scripts attached. The **When** script is executed just before the conversation node is to be displayed, and decides whether the node should be visible or hidden from the conversation. The author uses **When** scripts to decide at which point in the story the node should appear. The **What** script is executed just after the conversation node is displayed. Designers use the **What** script to execute game actions or change

game state at particular points in the conversation.

For large conversations, there may be conversation nodes that require the same script, or a slight variation on the same script. For each conversation node, the author must:

1. Determine what functionality is needed.
2. Search for existing scripts that provide that functionality.
3. If no such script exists, create a new script.
4. Attach the script to the conversation node.

It is possible that the desired script already exists, but the author needs to change a few parameters to accommodate the conversation node. For example, the toolset, as one of its default scripts, includes an "Intelligence" script that returns *TRUE* if the PC's intelligence is greater than 9. The author may want to change the number to 11 for a specific PC conversation node. Unfortunately the Aurora toolset does not allow parameters to be passed to a script.[2] Consequently, the author must create a new script to check if the PC's intelligence is greater than 11.

Even if the same script can be used for several conversation nodes, the author must still attach the script manually for each node. For example, the Emernick conversation only uses 8 unique **When** scripts (3 default, 5 custom), but those scripts are attached to 43 conversation nodes. The author also needs to know of the existence of scripts and what nodes where they will be attached. In Aurora there is no tool or mechanism to manage the intent or purpose of scripts other then their names. For example, the Emernick conversation uses a **When** script named "m1q2_emernik2", which suggests nothing of the script's intent other than it is used in the Emernick conversation.

When scripts on NPC conversation nodes can be particularly confusing. As described in Section 1.4.3, **When** scripts on sibling NPC conversation nodes are evaluated sequentially. This is analogous to the short-circuit semantics for boolean operators present in programming languages such as C, C++, and Java. If a script returns TRUE, all subsequent scripts are ignored. The majority of these **When** scripts can be complicated, involving many variables that are designated for controlling the plot and quests. Due to the short-circuit semantics, it can be difficult to determine which of the NPC sibling nodes will be displayed given certain conditions. For example, the "m1q2_emernik2" script – which is attached to an NPC node – has the following code:

```
int StartingConditional()
{
    int bCondition = GetIsPC(GetPCSpeaker()) &&
                     GetDistanceToObject(GetNearestObjectByTag("Emernik_Waypoint")) < 3.0 &&
                     GetLocked(GetNearestObjectByTag("Emernik_Door")) == FALSE ;
    return bCondition;
}
```

Neither the script's name or the code clearly reveals the intent of the script. This script only displays the NPC node if the NPC is inside a safe-room and the PC has securely locked the door to this

[2] At the time of writing, the sequel Neverwinter Nights 2 was released. It improves the toolset by allowing parameters to be passed to scripts.

room to prevent hostile creatures from entering. Additionally, the NPC node will not be displayed if the quest that involves Emernick is completed. However, the above code gives no indication that this condition is necessary. The reason is that another script, "m1q2_plotdone", is attached to an earlier sibling NPC node and consequently is always evaluated before the "m1q2_emernik2" script. If the Emernick quest is completed, the "m1q2_plotdone" evaluates to *TRUE* and the second "m1q2_emernik2" script is not even evaluated. The author now must consider the intent of both scripts when trying to determine under what conditions the NPC node will be displayed or hidden.

Lastly, many large conversations have 40 or more conversation nodes with scripts attached. It becomes difficult for the author to keep track of which nodes have scripts, and the intent of these scripts. Many scripts set local variables on objects so that other scripts on that and other conversations can check these variables in their **When** scripts. These *script interactions* require the author to mentally manage the structure of conversations, and complicates the process of fixing bugs when a conversation functions incorrectly.

1.6 Summary

This chapter introduced computer role playing games (CRPGs) and their unique characteristics as interactive stories from other genres of computer and console games. Neverwinter Nights was introduced as an example of a modern CRPG. The game's Aurora conversation editor was described as a powerful tool that allows game designers and players alike to construct their own stories. The editor allows authors to construct four primary elements, the world terrain, game objects, conversation trees, and scripts. The world terrain and game objects are constructed with a CAD-like interface. The conversations are built as a tree of nodes using the Aurora conversation editor. The scripts are written using NWNScript which has a C-like syntax. The Aurora toolset's deficiencies were then identified. The Aurora conversation editor does not abstract complex conversations in a way that is concise or can be grasped quickly. The manual scripting problem was also identified. Many authors do not have programming experience and scripting functionality by hand is an obstacle and a bottleneck in time.

This dissertation contributes a new conversation model that addresses these deficiencies through the use of design patterns. Chapter 2 describes how generative design patterns are used to replace manual scripting using the ScriptEase tool. Chapter 3 describes the structural components of the new conversation model. Chapter 4 describes the dialogue patterns that integrate with the ScriptEase model to replace the manual scripting of conversations. Chapter 5 describes the operations the author would use to build a conversation using the new model. The model is evaluated in Chapter 6 with a case study that compares the new model against the model used by the Aurora conversation editor. Finally, Chapter 7 discusses future work and concludes this dissertation.

Chapter 2

ScriptEase

2.1 Design Patterns

Unfortunately, writing scripts manually is the state-of-the-art in building computer role-playing games. However, attempts are being made to assist designers and programmers with scripting. Tools such as Epic's Kismet for the Unreal3 engine and Lilac Soul's Script Generator for Neverwinter Nights [13] remove some of the manual scripting burden. Instead of waiting for programmer assistance to write scripts, the author can use these tools to create scripts for them. Tools can come in different forms. For example, Kismet has a flowchart-like interface where the author can connect pieces of functionality together graphically. Lilac Soul's Script Generator has a wizard-like interface which interacts with the author through a series of questions, and generating a script at the end.

In general, CRPG scripts tend to repeat the same specific functionality with small changes in parameters. It is possible to represent these groups of similar scripts by a *design pattern*. Each design pattern shares the same overall code with specific components that are customized for each desired script. Design patterns are used extensively in Software Engineering for representing sets of design solutions for particular uses [7]. A pattern is a family of solutions that is then adapted to a specific solution instance by an author or programmer. Patterns are proven, robust solutions with little chance of the user making an error. Reusing patterns can greatly speed up development time, and reduces testing and debugging time.

There are two primary types of patterns: descriptive patterns and generative patterns. Descriptive patterns are the primary type of pattern described in the software engineering literature. A descriptive pattern describes a family of solutions in a neutral format. The programmer or author is then responsible of implementing a specific solution instance by manually writing code that implements the pattern description. For instance, Gamma *et al.* provide many descriptive patterns for programming in Object-Oriented programming languages [7]. They have a specific format to describe each pattern, including motivation, structure of the pattern, when to use it, and sample code in one or two programming languages.

Descriptive patterns greatly reduce the chance of the programmer making an error when writing

a solution by solving the general problem. However, the programmer or author can still introduce errors when implementing a specific solution instance of the pattern. Generative patterns solve this problem by not only describing a solution, but also generating code for that solution. The author adapts the pattern to a specific solution. A tool uses the adapted pattern to generate all the code necessary to implement that specific solution.

Generative patterns, although promising, may work poorly in general domains. The generated code may have poor performance. If patterns become too general, they require extensive adaptation before they can be used. However, under restricted domains, generative patterns can be quite effective. For example, in parallel programing CO_2P_3S (Correct Object-Oriented Pattern-Based Programming System) [20] uses generative patterns to help programmers generate a correct framework for parallel programs. The programmer first selects a pattern, which represents a parallel programming strategy (e.g. mesh, pipeline, etc.). The programmer then selects options to adapt the pattern to a particular application and then the code is generated for the framework. The programmer can then make application-specific changes at key points in the framework without having to worry about program correctness such as synchronization, for example.

Computer role-playing games are another domain suited for generative design patterns. Many scripts have a simple structure that can be represented as a pattern. For example, the author may want to spawn a guardian creature when the PC steals some items from a treasure chest. This can be considered a pattern called *Placeable Disturb - Spawn Creature*. This pattern can be used with any placeable objects, normally chests. The author then has to adapt the pattern to the specific treasure chest and guardian creature. Both the container and creature are considered options of the pattern, and are the pieces of the pattern that can be adapted. Once the options are specified, the code for the specific solution can be generated.

2.2 The ScriptEase Tool

ScriptEase is a tool to create and use generative design patterns for computer role-playing games [15]. Although it currently targets Neverwinter Nights, it is possible to port it to other role-playing games such as Oblivion. Using ScriptEase, an author can instantiate a new pattern, set some options to include specific objects in a module, add actions or conditions specific to the story context. The author can select any number of patterns from a pattern catalog. A pattern has a set of options, or parameters. Each option has a certain type, such as an integer, text, or game object. The author customizes a pattern by setting these options, either by providing a literal value for integers and string, or by using a picker to select objects. Only valid objects of the proper type are available in the picker. This prevents the author from making mistakes when selecting an object.

For example, the pattern *Placeable Disturb - Spawn Creature*, described in the previous section, has 3 options: **The Container**, **Creature Blueprint**, and **Spawn Effect**. Both **The Container** (e.g. a chest) and **Creature Blueprint** (e.g. a dragon) options require a game object, which can

be selected from the object picker. The third option, **Spawn Effect**, requires a visual effect to be displayed when the creature appears. This option can be selected from an enumerated drop-down list. In addition, the author may also want add an action to the pattern to make the creature speak some text upon spawning. Finally, the author can add a condition to the pattern to make the creature spawn if only a specific item is removed from **The Container** object. Adding actions, conditions and definitions will be described in Section 2.3.

Four types of patterns have been identified for computer role-playing games: Encounter, Behavior, Plot, and Dialogue patterns.

Encounter patterns support interactions between the PC and game objects, such as placeables, triggers, and doors. They generalize the encounter object found in the Aurora toolset. A typical example is the *Placeable Disturb - Spawn Creature* pattern just described. Encounter patterns were the first type of pattern to be implemented in ScriptEase[16].

Behavior patterns give life to NPCs, allowing them to perform actions and move in a believable fashion. For example, a guard patrolling a treasure chest containing a valuable item is a specific behavior. The NPC guard will guard the chest regardless of the PC's presence, but if the PC tries to intervene, the guard will defend the treasure. The guard example illustrates that most behaviors are ambient, and will run autonomously. Behavior patterns were the second type of pattern to be implemented in ScriptEase[3].

Plot patterns describe quests that occur frequently, allowing the author to control the game's story through these quests. The patterns include updating the PC's quest log at key points of the quest, as well as rewarding the PC with gold or experience when the quest is completed. The quest also controls whether parts of a story are available at a given point in time. A typical plot pattern involves an NPC asking the player to retrieve an item. The player must find this item in the game world and return it to the NPC for a reward. Currently quests are modeled using plot tokens, which can be assigned to objects to keep track of various states of a quest. There is ongoing work to promote plot patterns as first class objects in ScriptEase.

Dialogue patterns allow the author to build common structures found in conversations and to attach scripts to these conversations. For example, an NPC might react differently depending on the PC's charisma ability score. The author can include an *Ability* decision dialogue pattern to adjust the NPC's greeting according to the PC's charisma. Decision patterns are described in Chapter 4.

At the start of this research, ScriptEase had only one dialogue pattern: *Conversation When/What*. This general pattern allows the author to select a single conversation node from a conversation tree and change its **When** and **What** functionality. As described in Section 1.4.3, a **When** script determines if the conversation node will be displayed in the conversation, while the **What** script provides actions when that node is actually displayed in the conversation. Since the **When** part of the *Conversation When/What* pattern only has a condition placeholder, a condition must always be added. Similarly, the **What** part includes only an action placeholder and to be useful more specific

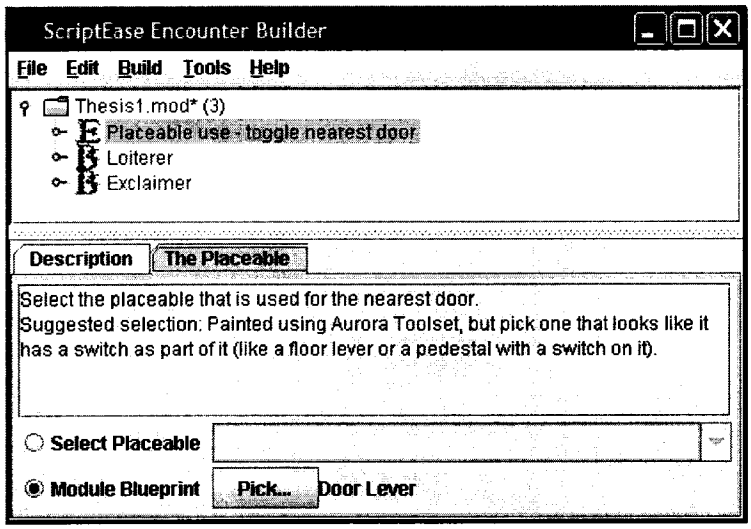


Figure 2.1: Selecting an Encounter pattern in ScriptEase.

actions must be added. For example, the author might adapt the **When** part of the pattern by adding a condition to check if the PC has high charisma. Next, the author could adapt the **What** part to move the NPC to a nearby game object.

Since *Conversation When/What* patterns are applied to a single conversation node, they do not identify useful conversation patterns. This research has identified numerous specialized dialogue patterns and has devised a new model to represent these patterns as well as a new set of building blocks to support the model. This model is described in Chapters 3 and 4.

2.3 ScriptEase Interface

ScriptEase has two tools: the Builder and the Designer. The Builder allows the user to instantiate patterns for a specific module. The Designer interface allows an author to create new patterns using the building blocks provided by the tool. If the author has programming experience, the most basic of these building blocks (atoms) can also be created with snippets of NWScript code.

To use ScriptEase, the author first loads a module that already contains game objects. Figure 2.1 shows the Builder interface with a module already containing some encounter and behavior patterns. The *Placeable use - toggle nearest door* pattern is currently selected, and the associated options are visible in the bottom half of the interface. The **The Placeable** parameter is visible, with the *Door Lever* placeable selected. In the game, the *Door Level* will open a nearby door when it is clicked on by the player.

The author can instantiate a pattern using the available pattern catalog. Figure 2.2 shows the ScriptEase encounter pattern instantiation interface. To instantiate a pattern, the author selects the desired pattern from the list on the right-hand side and the object to which the pattern applies from the list on the left-hand side. In this case the pattern is *Conversation When/What* and the object is a

conversation node. The first option of the pattern is the object to which the script is attached after the code is generated. The author can select the pattern or object in either order. If the object is selected first, only a subset of patterns that can have the selected object as a valid first option are available. The *Conversation When/What* pattern has only one option: a conversation node. The the middle of the screen in Figure 2.2 the conversation node picker with a conversation node selected. In this case, the author is going to make the node appear only if the PC is female, since Earl the bartender will go out of his way to get drinks for female patrons. When the node is displayed in conversation, Earl will move twice: once to the cabinet and once to return back to his original location.

Figure 2.3 shows the Builder interface with the instantiated dialogue pattern. The bottom half of the interface shows the conversation node option that was set during instantiation. At this point, the pattern is considered complete and code can be generated. However, the *Conversation When/What* pattern was constructed to be general, and therefor the generated scripts will do nothing. The pattern must be further adapted to suit the author's idea. To do that, the author opens the pattern to view the internal components as shown in Figure 2.4.

At the top level there are two *Situations* marked with the stylistic **S**. A situation responds to a single event since a pattern can encapsulate code for several events. The *Conversation When/What* pattern has two situations: one for the **When** event, and one for the **What** event. Situations can contain definitions, conditions and actions, each of which can have options. During code generation, each component generates scripting code. The first component of any situation is an *Event*, marked with a **V**. A ScriptEase event represents a specific game engine event that occurs in Neverwinter Nights. The pattern provides a scope, allowing components to use options specified at the pattern level.

In Figure 2.5 the author adapts the **When** situation to add a *Definition*, marked by a **D**. A definition allow the author to add game state into the pattern so that other components can use it. In Figure 2.5, the author adds a boolean definition called "Is Specific Gender" to check if the PC has the female gender. Additionally, event atoms often contain implied definitions, which provide game state specific to the event. For example, in Figure 2.6 the **When** event has an "NPC Speaker" definition and "PC Speaker" definition. These definitions allow the author to refer to the two conversationalists throughout the pattern. The author uses the implicit "PC Speaker" definition when constructing the "Is Specific Gender" definition.

In the context of the pattern the author can specify the condition for the **When** script that allows the conversation node to be displayed. In Figure 2.7 the placeholder condition has been replaced with a **Condition** marked with a **C**. A condition takes a binary option, which determines if the condition is true. In this case the author has used the "Is Specific Gender" definition as the condition's option. Now the conversation node will only be displayed in the conversation if the PC is female.

With the "When" situation successfully adapted, the author can now adapt the "What" situation. Figure 2.8 shows the author adding an Action atom, marked with an **A**, to the situation. An Action

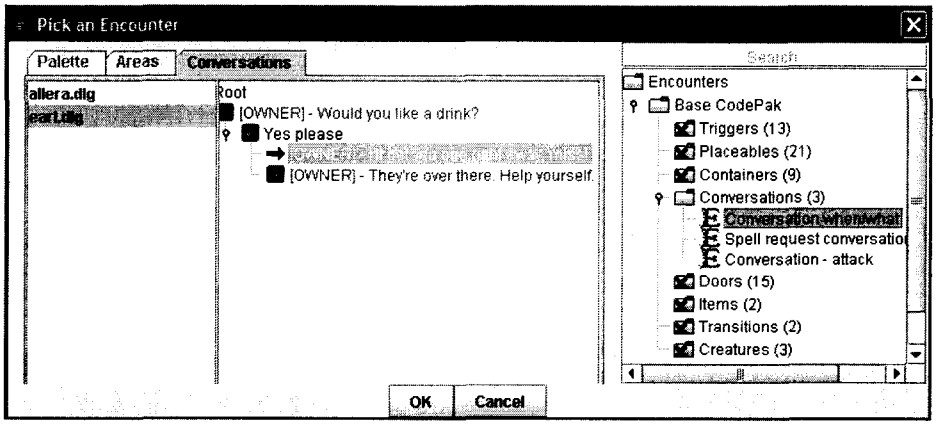


Figure 2.2: Creating a new pattern instance.

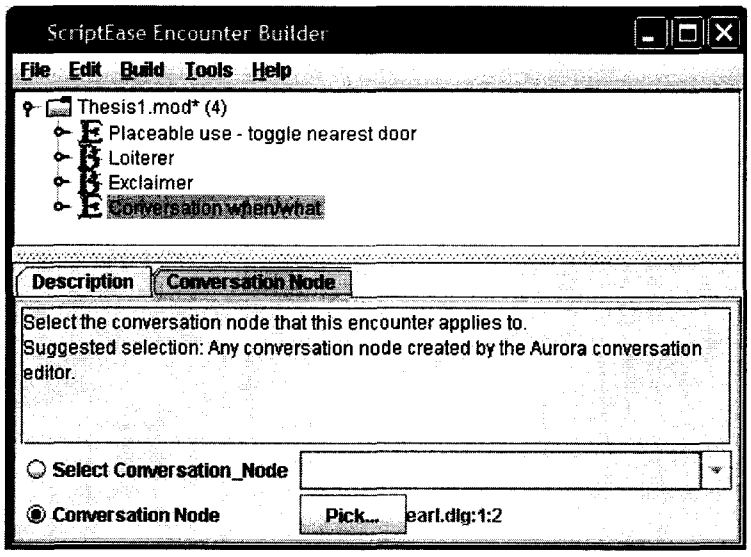


Figure 2.3: Creating a new pattern instance.

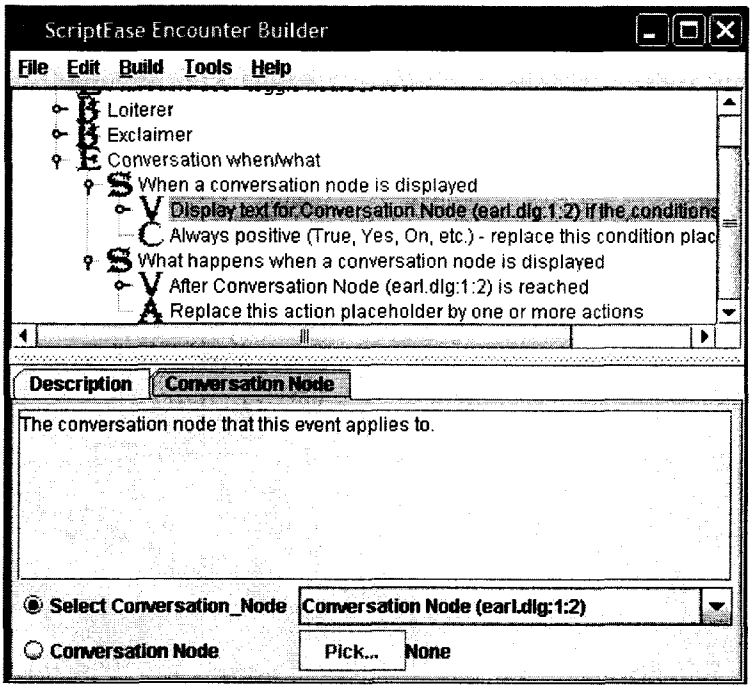


Figure 2.4: The internals of the Conversation When/What pattern.

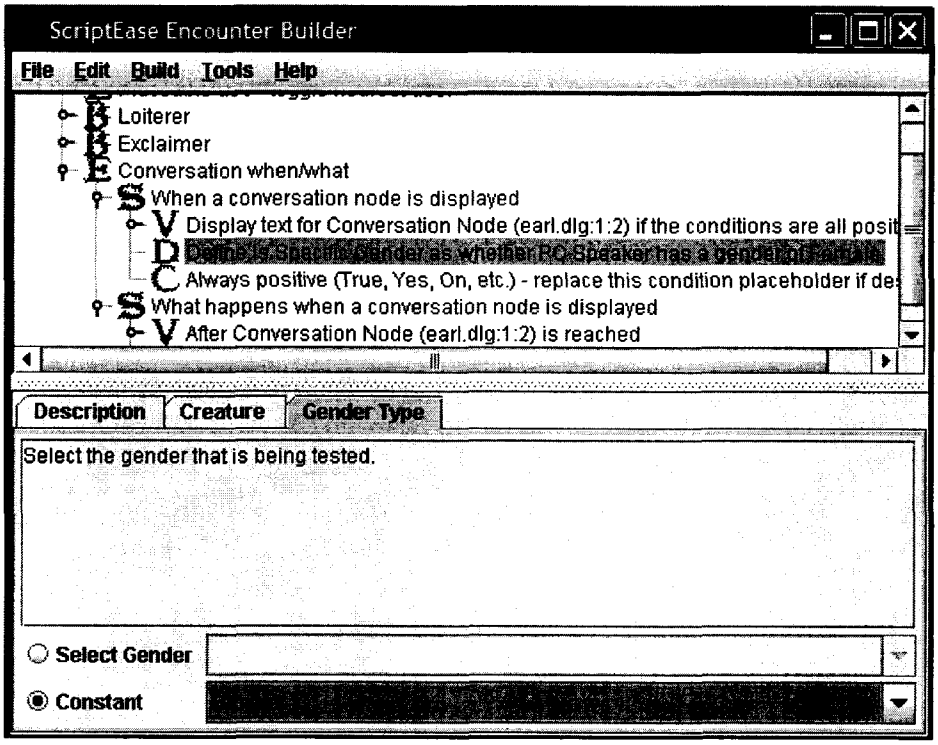


Figure 2.5: Adding a definition atom that checks the PC's gender.

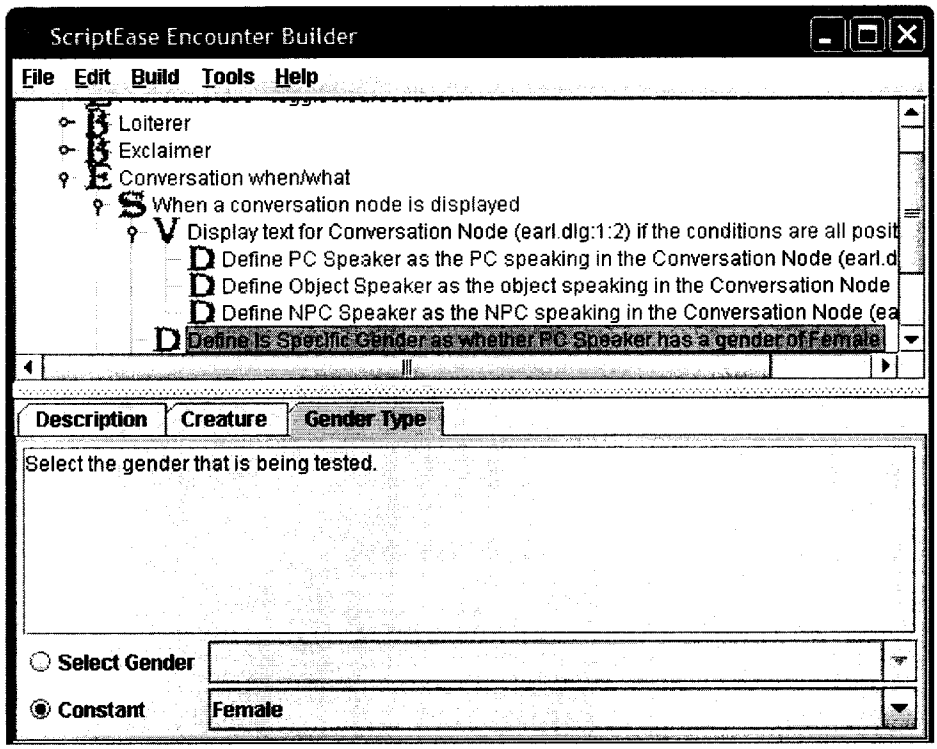


Figure 2.6: An event's implied definitions.

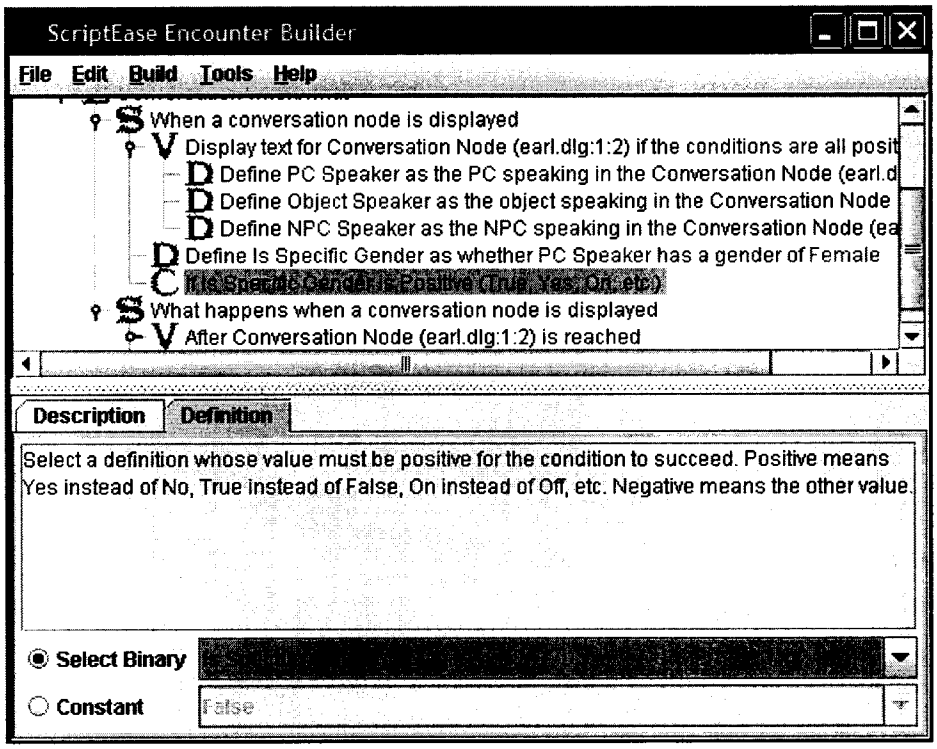


Figure 2.7: A positive condition that will return true if Is Specific Gender is true.

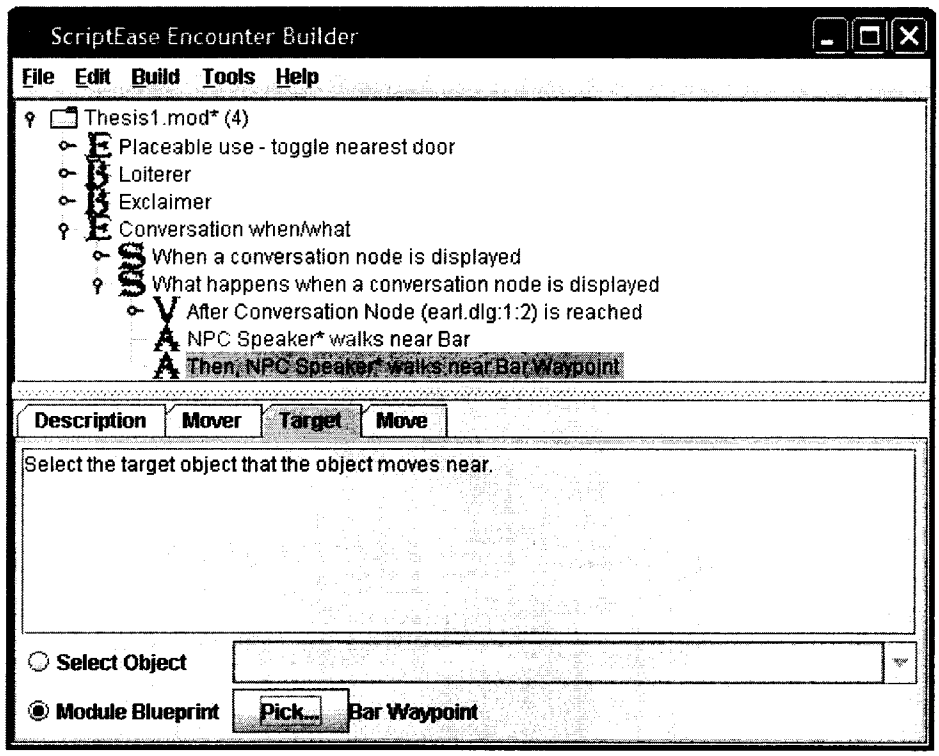


Figure 2.8: Two action atoms that move the NPC towards objects.

atom represents a single logical action that can occur in the game. In Figure 2.8, the added action forces the NPC to walk to the bar to simulate getting a drink. Next, the NPC walks back to his original location marked by a waypoint. With these actions, the pattern is now fully adapted, the code can be generated by the tool, and the author can test the pattern by playing the module.

The above example demonstrates several types of adaptation. Previous work has identified these adaptations and classified them into levels of cognitive difficulty [2]. The Table 2.1 shows the nine possible adaptations that can be performed on a pattern. They are ranked in increasing cognitive difficulty, with setting options as the easiest and only necessary adaption, to adding new situations which requires selecting the event and adding conditions, definitions, and actions.

The Designer is similar to the Builder, as shown in Figure 2.9. An author can construct new patterns using the same nine operations that are used to adapt existing patterns. There is a seamless transition from using patterns to creating patterns. Furthermore, new atoms (actions, definitions, and conditions) can be created by adding fragments of NWScript code. Figure 2.9 shows a new action atom, with the NWScript code in the bottom half of the window. The atom's options are exposed as function arguments, allowing the NWScript to use those options as variables.

Cognitive Level	Adaptation
1	Set Pattern Options
2	Delete a Situation
3	Delete an action/definition
4	Delete a condition
5	Replace an action/definition placeholder
6	Add an action/definition
7	Replace a condition placeholder
8	Add a condition
9	Add a situation

Table 2.1: Cognitive levels of pattern adaptation.

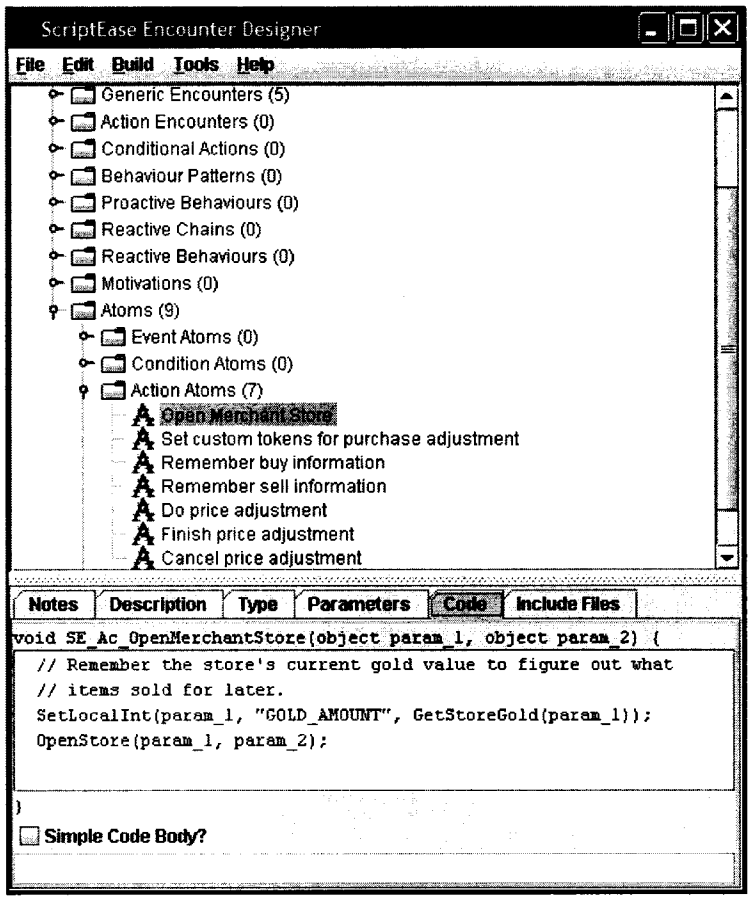


Figure 2.9: A custom atom for merchant objects created in the ScriptEase Designer.

2.4 Summary

This chapter introduced the concept of generative design patterns as a solution to the manual scripting problem found in state-of-the-art CRPGs. The ScriptEase tool was then described as an example of how generative design patterns can generate scripts for Neverwinter Nights. An author can use a pattern by setting options and further adapt the pattern by adding or removing actions, definitions, and conditions. Fully adapted patterns can then generate all the necessary scripting code without any author intervention. Finally, the ScriptEase tool was described in detail, including descriptions of the tool's interface and pattern components.

Chapter 3

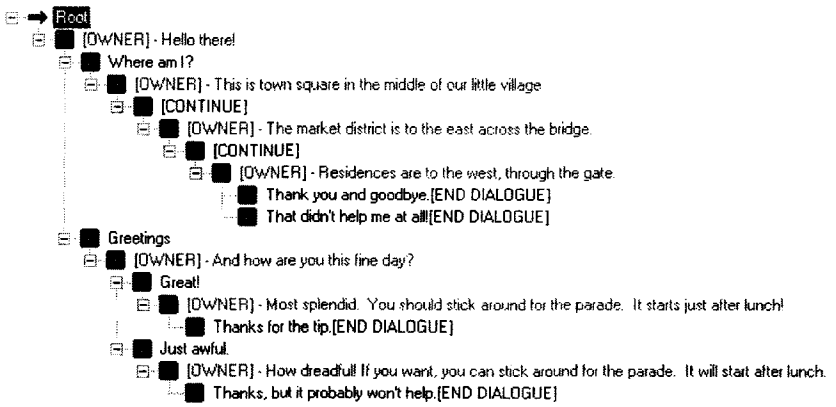
Structural Patterns

This dissertation presents a new model that provides abstraction mechanisms for constructing and viewing conversations. The notion of manually creating and attaching NWScript scripts in conversation nodes is replaced with generative design patterns. Dialogue design patterns are different from encounter, behaviour, and plot patterns currently in ScriptEase. Dialogue patterns are more structure-based than intent-based, and thus cannot be prototyped with current ScriptEase pattern components. Nevertheless, the model can be integrated with the ScriptEase tool and its existing patterns. At the time of writing, the model has not been implemented into a functional conversation editor in ScriptEase, but there is future work to do so.

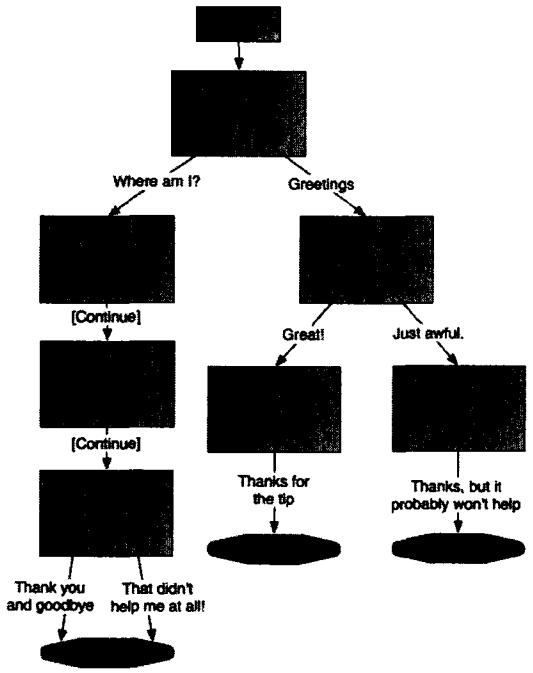
The presented model addresses the disadvantages of the Aurora conversation editor outlined in Section 1.5. Note that all figures provided to illustrate the model are symbolic and might not resemble an actual graphical user interface (GUI). However, a possible implemented GUI may have similarities in appearance. This chapter presents the model in a bottom up manner by introducing structural patterns. Chapter 4 expands on the model by describing decision patterns and optional choice patterns.

Structural patterns represent commonly occurring conversation trees. They encapsulate the structure of the tree and non-scripting properties of conversation nodes such as remark text, sound files, and animations. An author can construct a conversation tree by building up the components directly, or by combining existing structural patterns together and adapting them using basic component operations. Since a structural pattern is a conversation tree, it can be used as a sub-tree in other patterns. Consequently, structural patterns can be composed with each other to build larger structural patterns. This is different from the intent-based patterns in ScriptEase. For example, there is no way to compose two Encounter patterns into a larger Encounter pattern.

Structural patterns contain 4 basic components: *exchanges*, *topics*, *link* and *end dialogue targets*, and *topic groups*. Each will be describe in turn.



(a) A friendly conversation in the Aurora conversation editor.



(b) A friendly conversation represented with exchanges.

Figure 3.1: Two representations of a simple conversation.

3.1 Exchanges

One of the problems Section 1.5 identifies with the Aurora conversation editor is how sibling nodes move spatially further apart as lower levels of the tree are exposed. It quickly becomes impossible to associate sibling nodes with their parent node without a significant amount of scrolling on the screen, at which point the author cannot get an overview of the conversation's structure.

Dialogue patterns tightly couple a parent NPC conversation node with its PC children into a new construct called an *Exchange*. In a diagram, such as the ones shown in Figure 3.1(b), an exchange is a red coloured box which represents an NPC node[1]. Inside the box are blue circles called *choices*, each representing a PC child node. This representation closely approximates what a player would see as a conversation when playing the game. A conversation window in Neverwinter Nights always has an NPC remark followed by one or more choices, i.e. a single exchange. An exchange contains a single *NPC remark*. A remark has the same properties as a conversation node in the Aurora conversation editor, such as text (remark text), animation, and a sound file. Similarly, a choice contains a *PC remark*. Remarks can be used as options for ScriptEase patterns.

Exchanges can be connected by drawing an arc from a choice to the top of another exchange. Figure 3.1 illustrates how a conversation – termed the friendly conversation – would look both in the Aurora toolset and as dialogue patterns. Figure 3.1(a) shows the conversation in the Aurora conversation editor. Figure 3.1(b) shows the same conversation as exchanges. Notice that exchange boxes hold the remark text for NPC nodes, and the arcs hold the remark text for PC nodes; this keeps the nodes compact and visible. The exchange version of the conversation also features *End Dialogue targets*. These will be explained in Section 3.3.1.

The conversation in Figure 3.1(b) highlights some of the advantages of exchanges. The first exchange, labeled "Hello there!" has two choices "Where am I?" and "Greetings". Both choices connect to a sub-tree composed of several exchanges. Notice that regardless of the size of the sub-tree, even if the sub-trees were composed of tens or hundreds of exchanges, the two choices still remain spatially close together inside the first exchange. In a GUI, the choice labels can be rendered close enough to the choices that they will not become separated as the conversation tree below grows in size.

In Neverwinter Nights it is possible for NPCs to speak one-liners. These are single conversation remarks that appear above the NPC's head instead of opening a conversation window. The game stores these one-liners inside conversation files, just like other conversations. Aurora considers one-liners special, as they are direct children of the conversation's featureless *Root* node, and have no child nodes or end of dialogue indicators. In the Dialogue Pattern model, these one-liners are called *Utterances*. *Utterances* are exchanges that have no choices. Similar to the Aurora conversation editor, utterances can only be children of the *Root* node.

[1] In the Aurora conversation editor, NPC nodes are also coloured red. PC nodes are coloured blue.

3.2 Topics

In complex conversations, such as ones in the NWN campaign story, an NPC may have a lengthy explanation for the PC. Instead of putting all the text into a single NPC node, it is common practice to split the explanation across multiple NPC nodes. This reduces the amount of text the player has to read at any one point in the conversation. For example, in Figure 3.1(b) after the "Where am I?" remark, the NPC's explanation is spread across three exchanges. For the first two exchanges, the PC only has one choice: continue. In the third exchange, the NPC finishes the explanation and the PC has more choices available. These three exchanges are identified as a *Topic*, where each exchange holds a piece of explanation for that topic.

Formally, a topic is defined as:

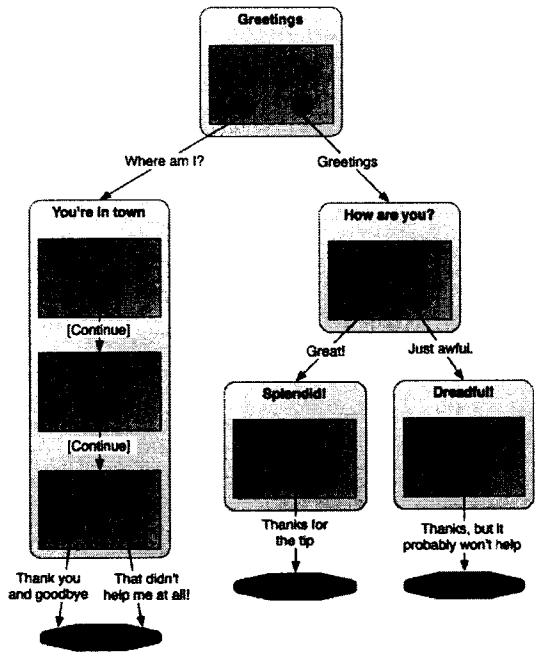
1. One or more exchanges where,
2. All exchanges except for the last exchange must have exactly one choice that connects to the next exchange in the topic, and,
3. The last exchange in the topic may have one or more choices.

By this definition, all exchanges in a conversation are considered a part of a topic. Any exchange with two or more choices that has a parent exchange that also has two or more choices must be a topic of length one. For example, in Figure 3.1(b) the exchange with the "And how are you this fine day?" remark would be considered a topic of length one.

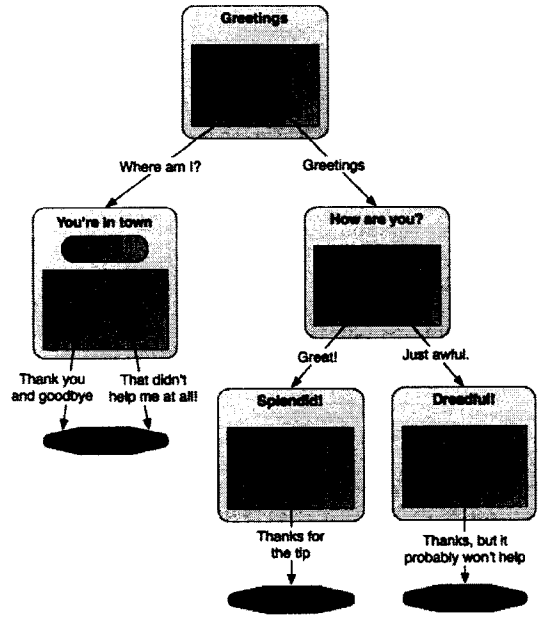
The complexity of a conversation can be reduced by encompassing exchanges into topics. Figure 3.2 shows the friendly conversation with topics introduced. Each exchange is surrounded by a second box representing the topic. Notice the three exchanges after the "Where am I?" choice are in a single topic. The first two exchanges are called *inner exchanges*. The third exchange is called the *tail exchange*. Each topic has exactly one tail exchange and have zero or more inner exchanges that proceed the tail exchange.

With each exchange now encapsulated into a topic, the conversation can be further abstracted. Topics can be collapsed by hiding all inner exchanges in the topic with the tail exchange visible. In a collapsed state, a number is placed above the tail exchange to indicate the total number of exchanges in the topic. For example, Figure 3.2(b) shows the friendly conversation with a collapsed topic. The conversation complexity has been reduced without losing important information, such as the conversation's branching structure. A GUI could allow the author to expand and collapse a topic as more or less information is needed.

When a topic is collapsed, information is lost. Specifically, the author can no longer see the remark text of the hidden exchanges. To counteract this, topics also introduce a new feature: *Topic Intents*. The topic intent is text that the author can use to quickly summarize the NPC remarks in the topic. In the Aurora conversation editor, the author must scan the full remark text to understand what



(a) A friendly conversation with an expanded topic.



(b) A friendly conversation with a collapsed topic.

Figure 3.2: A friendly conversation with topics added.

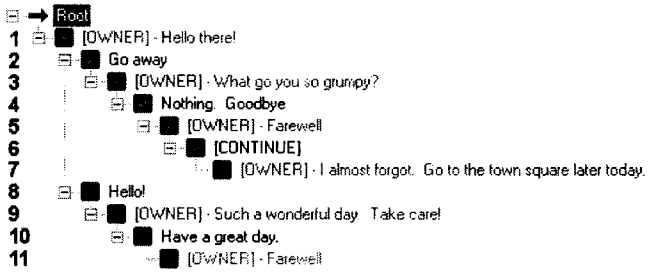
the NPC is saying. Sometimes, these remarks can be extremely verbose, with the intent found either in the middle or at the end of the explanation. Including topic intents allows the author to quickly scan the conversation to find a particular topic. For example, in Figure 3.2(b) all three exchanges in the collapsed "You're in town" topic discuss the layout of the town. The author can understand the intent of the topic without having to expand the topic and read the individual exchanges.

3.3 Link Targets

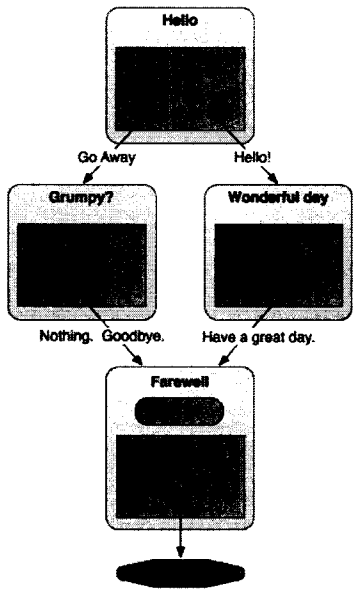
Section 1.4.3 describes how the Aurora conversation editor allows the author to create link nodes that point to other nodes in the conversation tree, which converts the conversation from a tree into a graph. Instead of duplicating a sub-tree of nodes under two conversation nodes, the author can create a link node under the first node that points to the sub-tree found under the second node. For example, in Figure 3.3(a), the PC can respond to the NPC's greeting in two ways: "Go Away" and "Hello" (lines 2 and 8). In either case, the author wants the NPC to impart some common information at the end of the conversation. The author first creates the common "Farewell" (line 5) conversation sub-tree as part of the sub-tree rooted at "Go Away". Next, the author creates the sub-tree rooted at "Hello" and instead of creating a second "Farewell" sub-tree, the author creates a link node on line 11 that points to the "Farewell" node on line 5. Now regardless of the PC's choice in the conversation, the NPC will speak the farewell sub-tree. If a conversation node that is a destination of a link is deleted, then the link node is deleted as well.

Dialogue patterns also have a mechanism to support conversation sub-tree re-use. This is done by allowing multiple choices to point to the same topic. For example, Figure 3.3(b) shows the conversation in Figure 3.3(a) converted to the dialogue pattern model. Both the "Nothing. Goodbye." and "Have a great day" choices point to the "Farewell" topic which contains two exchanges. Both choices *directly link* to the "Farewell" topic. Direct links from choices to topics are analogous to link nodes that link to NPC nodes in the Aurora conversation editor. However there is a difference. With the Aurora conversation editor, one of the links is special in that the conversation nodes are textually embedded as a sub-tree under the link where as other links are link nodes. With dialogue patterns, all direct links have first class status. Note that choices directly link to topics and not exchanges. Sharing a choice between several exchanges is discussed in Section 3.4. If the author wants to directly link to an exchange that is in the middle of a topic, the topic can be split into two topics. Splitting a topic makes sense in this case since an entry point in the middle of a true topic is problematic.

Direct links are clear and easy to understand. There is only one representation of the destination topic, unlike linking in the Aurora conversation editor which requires special link nodes to act as placeholders for the sub-tree to which they are linking. The editor distinguishes between the first conversation node which holds the shared sub-tree, and the other nodes which have a link node pointing to the sub-tree. Unfortunately, in a GUI direct links only reduce visual complexity when

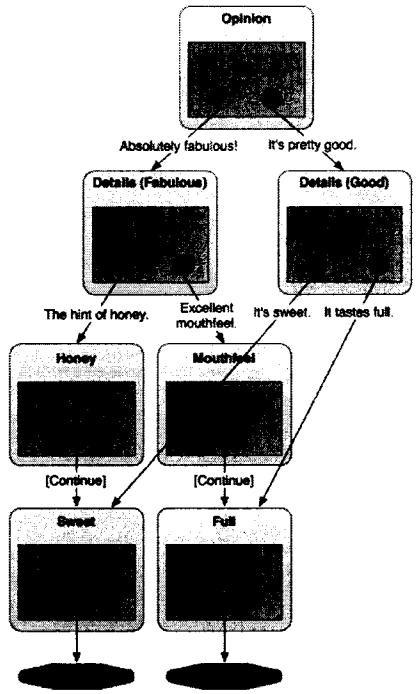


(a) A link node in the Aurora conversation editor.

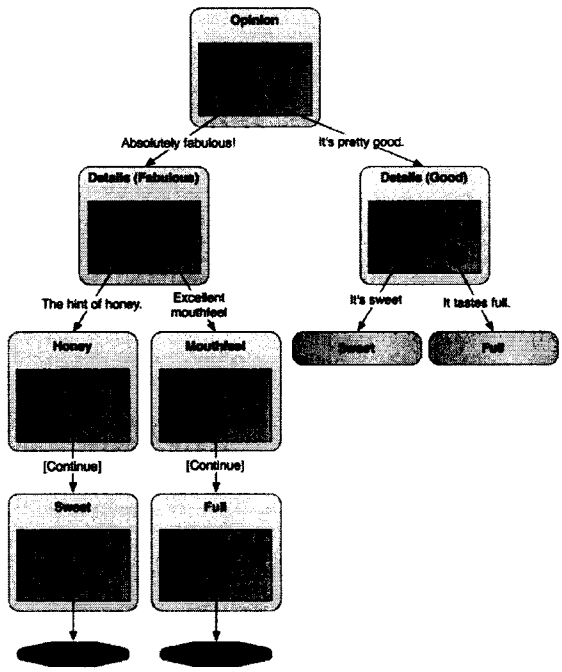


(b) A direct link in the Dialogue Pattern model.

Figure 3.3: A friendly conversation with links.



(a) Direct links crossing over other model components.



(b) Introducing link targets removes the interfering arcs.

Figure 3.4: Contrasting the visual complexity between direct links and link targets.

the choices are spatially close to the target topic. The choices and the target topic can also be far apart in the conversation tree. Direct links can increase the visual complexity if the arc from the choice to the topic intersects with other components of the conversation tree. For example, the wine conversation in Figure 3.4(a) has a "Details (Good)" topic with two choices that directly link to the "Sweet" and "Full" topics. Since the choices are separated from their destination topics, the connecting arcs intersect with the "Mouthfeel" topic. A possible solution is to draw the arcs with angles or curves to go around other components, but that increases the overall length of the arc making it difficult for the author to determine which choice links to each topic. It also complicates the GUI implementation since the program would have to compute a good layout to draw the arcs.

In these cases it is necessary to return to the link placeholder mechanism found in the Aurora conversation editor. A *link target* is used as a placeholder in the dialogue pattern model in a similar manner as a link node is used in the Aurora conversation editor. In the wine conversation in Figure 3.4(b), the offending direct links have been replaced with link targets. The link target is labeled by the same intent of the target topic. A link target remains spatially close to its corresponding choice which prevents any clutter when drawing conversation components. With link targets, there are now two representations of a link. A *direct link* is where a choice directly links to the topic. A *secondary link* is where a choice points to a link target representing the topic. For example, in Figure 3.4(b) the "[Continue]" choice in the "Honey" topic has a direct link to the "Sweet" topic. The "It's sweet" choice in the "Details (Good)" topic has a secondary link to the "Sweet" topic. It is possible for there to be more than one direct link to a topic depending on how a GUI chooses to render the conversation components. Secondary links also allow a choice to link to an ancestor topic. This would be difficult to do with direct links without intersecting components or arcs.

There are still advantages of this approach over the Aurora conversation link node approach. To counter a secondary link's extra level of indirection, a GUI could allow the author to double-click on the link target to redirect the screen viewport to the destination topic. This is analogous to the Aurora conversation editor's double-click mechanism on link nodes. For example, in Figure 3.5 the viewport is focused on the "Rush" topic where the "Other questions?" is a secondary link to its target topic, which is two screen-widths away. Instead of scrolling the screen to find the corresponding topic, the author can double-click the link target to re-position the viewport onto the topic.

However, an author may find shifting the screen viewport to be visually jarring and disorienting – as it is in the Aurora conversation editor. Another mechanism is to convert the secondary link to a direct link by moving the topic to the viewport. For example, Figure 3.6 shows the same conversation with the "Other Questions?" link target and its topic swapped. The topic is now in the author's view and no scrolling is needed. The conversation is still structurally identical to the conversation in Figure 3.5, only the visual representation has changed. This mechanism allows the author to view target topics in the context of the conversation. This redirection mechanism is superior to the double-click moving viewport mechanism in that any secondary link can become a

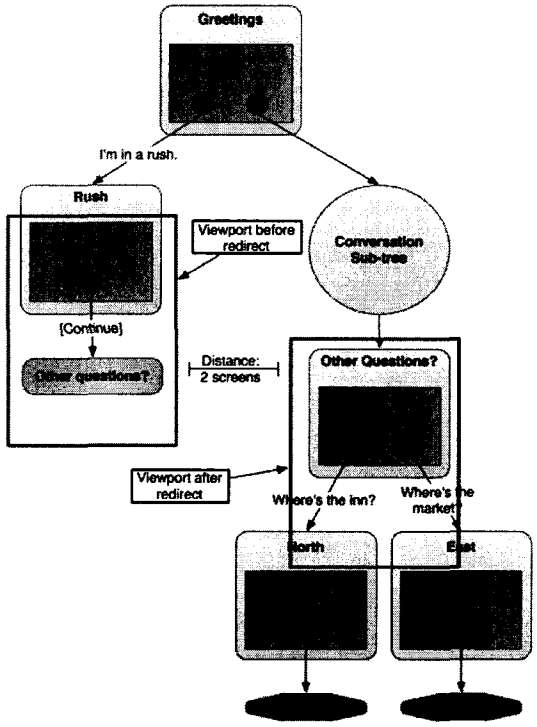


Figure 3.5: The redirect GUI operation moves the viewport to the target topic.

primary link so that its full context can be viewed. In addition, multiple direct links are possible in many situations and moving target topics to the author's view makes direct linking more likely.

3.3.1 End Dialogue Targets

An *End Dialogue target* is a special link target that indicates at which points in the tree the conversation will end. A choice that has an arc to an end dialogue target will end the conversation if selected by the PC. For example, in Figure 3.3(b), the choice in the "Farewell" topic is connected to an end dialogue target. End dialogue targets help the author to determine which choices end the conversation.

3.4 Topic Groups

In Section 3.3, direct and secondary links were introduced as mechanisms to link choices to topics. These links allow a topic to be shared by multiple choices and are analogous to link nodes that point to NPC nodes in the Aurora conversation editor. However, an author may want to share a choice across several exchanges. For example, the Aurora conversation in Figure 3.7(a) has several PC link nodes that point to the PC node "Goodbye" that is the root of a common sub-tree that ends the conversation. In Figure 3.7(b), the conversation is converted into a dialogue pattern. The same "Goodbye" choice is duplicated across all exchanges which requires the author to set the properties,

such as remark text, for each choice. These duplicated choices then link to the topic that ends the conversation.

Both examples are structurally equivalent, and both share the same disadvantage of duplicating the PC node or choice that the author wants to share. The Aurora example has terminal PC link nodes as duplicates whereas the dialogue pattern example has the choice duplicated in several exchanges, each with a link to the target topic. The duplication in dialogue patterns can be avoided with a more concise and compact representation by introducing *topic groups*. Topic groups are created using two separate mechanisms: *exchange customization* and *choice customization.*

3.4.1 Exchange Customization

Instead of duplicating a choice across several exchanges – each in a different topic – it is more concise to collapse all involved topics together into a single *topic group*. For example, in Figure 3.8, the topics that contained the duplicated "Goodbye." choice are now co-located as tabs in a single topic construct. Only the internals of one topic can be *visible* at any one time, and in Figure 3.8(a) the "Great" topic is visible with its exchange visible. The author can change the visible topic by clicking on a topic tab. In Figure 3.8(b) the "Dreadful" topic is now visible.

Since only certain exchanges are visible depending on the visible topic, collecting topics into a topic group is called *exchange customization*. The author *customizes* the exchange view by making a certain topic visible and making changes to the topic's exchanges. Exchange customization further abstracts the conversation tree by hiding more information, i.e. exchanges. The author can use the topic intents displayed in the tabs to get an overview of the conversation structure. Clicking on a topic tab makes the topic's interals visible, allowing the author to view detailed information such as NPC remark text. For example, in Figure 3.8(b) the "Dreadful" topic is now visible and the NPC remark text has changed. Choices can link to topics in a topic group by drawing an arc to the topic tab.

Exchange customization easily accommodates sharing a set of choices across several exchanges. The question-answer conversation in Figure 3.9(a) has three exchanges that have the same two choices. In Figure 3.9(b), the three exchanges have been co-located into a topic group with the two choices shared. In this example the exchanges share the same set of choices. However, exchange customization can be further generalized by allowing topics that only share a subset of choices to be grouped together. For example, in Figure 3.10, the "East Side" topic contains an extra choice not shared by the other topics. These topics can be converted into a topic group by including the union of all choices from all topics into the topic group. In Figure 3.11(a), the topic group contains the shared choices as well as the extra "How wide is the river?" choice. Even though this choice is not shared by all exchanges, it remains visible regardless of which topic is visible. This makes the choice's sub-tree visible at all times, which keeps the conversation tree structure independent of the visible topic. Otherwise, the author needs to manage visible topics to be able to view the entire

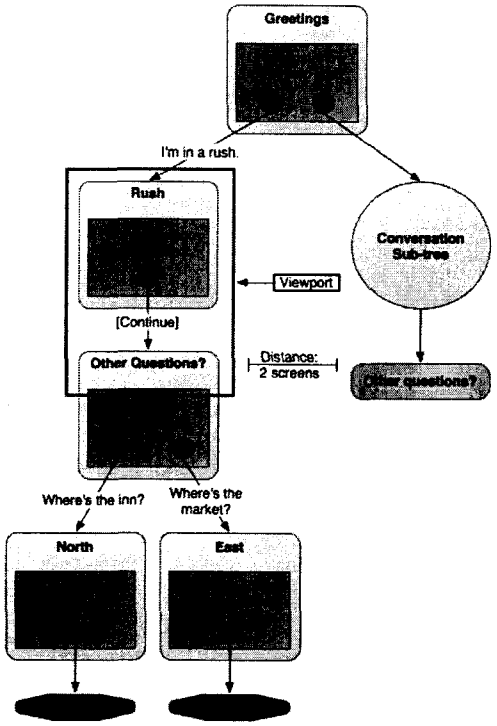
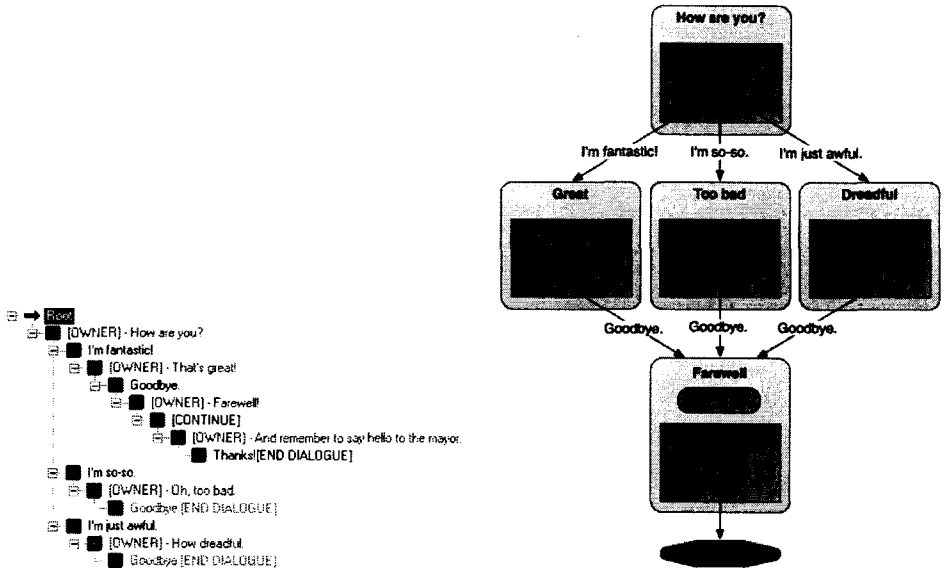


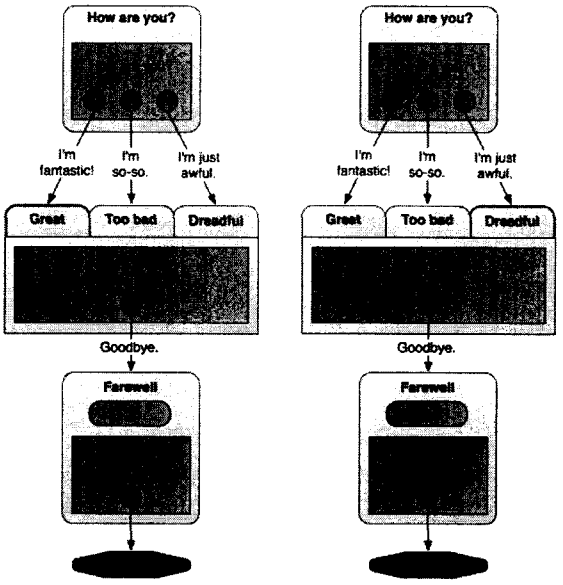
Figure 3.6: The swap GUI operation move the target topic and sub-tree to the viewport.



(a) The conversation with PC link nodes in Aurora.

(b) The conversation as a dialogue pattern with duplicated choices and direct links.

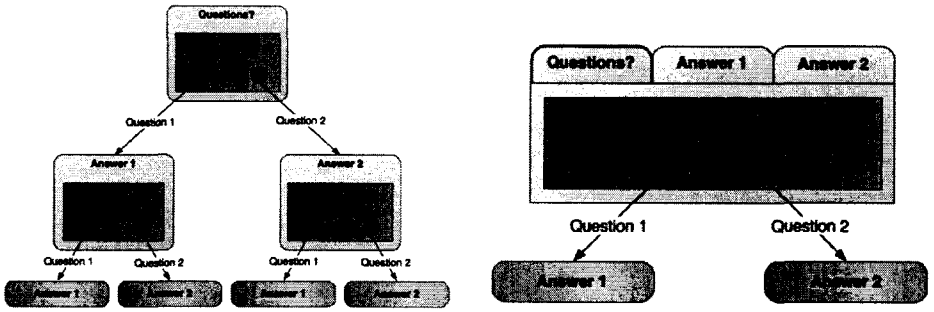
Figure 3.7: Converting the farewell conversation from Aurora with PC link nodes to a dialogue pattern.



(a) Topic group with the "Great" topic visible.

(b) Topic group with the "Dreadful" topic visible.

Figure 3.8: The farewell conversation with topic groups and only one choice.



(a) All three exchanges share the same choices.

(b) The three topics merged into tabs. Two exchanges are now hidden.

Figure 3.9: Dialogue pattern for answering questions in a conversation.

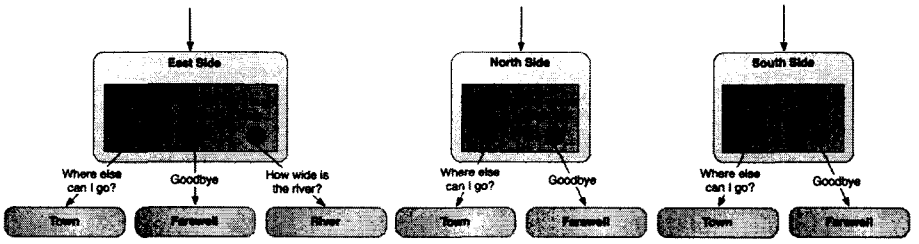
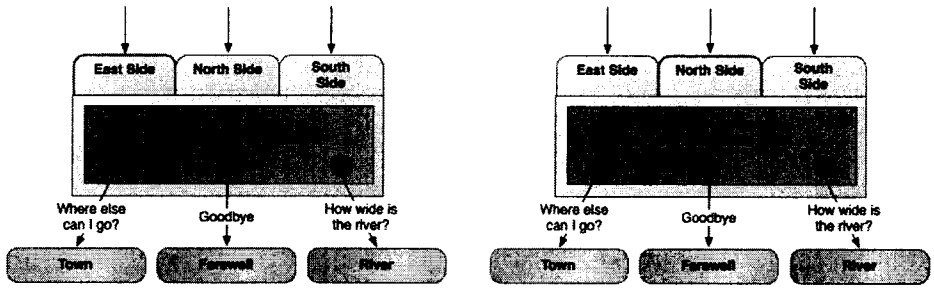


Figure 3.10: Three topics with duplicated choices. One topic has an extra choice.



(a) Three topics in a topic group. All three choices are enabled for the visible topic.

(b) Three topics in a topic group. One choice is disabled for the visible topic.

Figure 3.11: A topic group with a subset of shared choices.

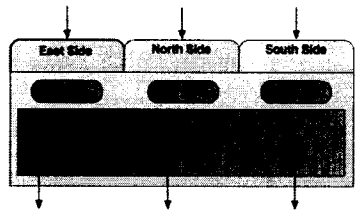


Figure 3.12: A topic group with topics that have a different number of exchanges.

conversation tree. Changing the visible topic should only change the view of the internal exchanges, not the external topology. Instead, choices can be marked as *disabled* with a different colouring to indicate they are not shared by the visible topic. For example, in Figure 3.11(b), the "North Side" topic is visible and since it does not share the "How wide is the river?" choice, the choice is disabled and coloured a dark blue. If the "East Side" topic becomes visible, the choice is no longer disabled and reverts back to the original light blue colour.

It is still possible for a topic in a topic group to have inner exchanges. In fact, different topics in the same topic group can contain one or more inner exchanges. If the topic is the visible topic, its inner exchanges can be expanded and collapsed as normal. A topic group with a topic that has inner exchanges has to indicate the number of exchanges in each topic. For example, in Figure 3.12, the "East Side" topic has two exchanges with the remaining topics each containing only a tail exchange. The numbers are always visible regardless of the visible topic, allowing the author to quickly determine the size of each topic. As a short-hand, if all topics in the topic group each contain only one exchange, then these number will not be shown in further diagrams.

The advantage of topic groups becomes apparent when converting complex conversations in Chapter 1 of the NWN official campaign into dialogue patterns. For example, the small subset of the NPC Bertrand's conversation in Figure 3.13 shows six topics in a topic group with five choices. The dark blue disabled choices indicate that not all choices are shared. If these topics were not in a group,

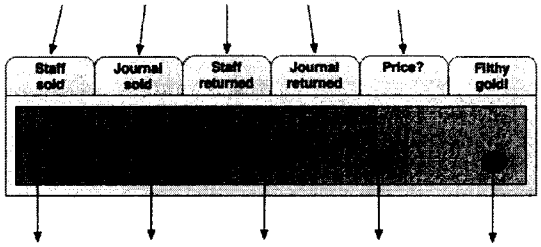


Figure 3.13: A piece of Bertrand's conversation. Several topics are in a topic group.

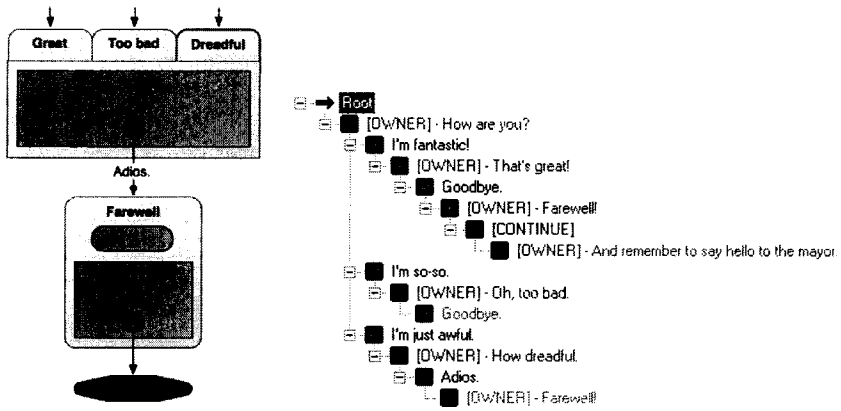
the five choices would have to be duplicated resulting in a total of 20 choices distributed across six topics. Each of these choices can potentially have a secondary links with an attached link target for a possible total of 20 link targets. The conversation tree would be cluttered and confusing with redundant information. Topic groups simplify this conversation sub-tree by removing 15 redundant choices and 15 potential link targets. In contrast, the same piece of conversation in the Aurora conversation editor would be composed of 26 conversation nodes with six NPC nodes, five PC nodes, and 15 link nodes.

3.4.2 Choice Customization

When topics are co-located into a topic group, the shared choices have identical PC remarks across all topics in the group. Sometimes an author may want to change or *customize* the PC remark text of a shared choice for a single topic in the group. For example, for the third topic "Dreadful" in Figure 3.8, the author may want to change the choice text to "Adios". To do this, the author could separate the "Dreadful" topic from the group, duplicate the choice, and then change the remark text. This is unnecessarily complex since the author has to create two new components, a new topic and a possible link target, to only change the remark text of a single choice. The Aurora conversation editor avoids this problem since link nodes can link to PC nodes and therefore no PC node duplication is necessary.

Instead, the author can use *choice customization*. The choice is marked as *customized* by an upper-case **C**. Analogous to using disabled choices, the **C** indicates to the author that the choice has different text for certain topics in the topic group. The choice's remark text will change as the visible topic changes. For example, in Figure 3.14(a), the choice has been customized, and the choice's remark text says "Adios" when the third topic "Dreadful" is visible. This compact representation can be contrasted to the Aurora conversation in Figure 3.14(b) where the author has to replace a link node that points to a PC node with a full-fledged PC node to change the remark text. The author then has to add a NPC link node to link to the shared sub-tree. This process increases conversation tree size by one node. To customize a second PC node, the author would repeat the procedure and the extra link node would again increase the size of the tree.

A choice customization affects only one topic in the topic group. The original choice properties



(a) A customized choice in a topic group.

(b) A customized PC node in the Aurora conversation editor.

Figure 3.14: Customizing a choice to say "Adios."

will remain shared with the remaining topics. The author can perform a second customization operation on a second topic if necessary. This customization is distinct from the first, and changing PC remark properties for one customization will not affect the other. The author can at any time remove a customization to revert the properties of a choice's PC remark to the original shared values.

3.5 Dialogue Generation

After an author has constructed a conversation using structural patterns, the system needs to convert the topics, exchanges, choices, and links back to the native conversation format recognized by Neverwinter Nights. This is analogous to encounter and behavior patterns generating NWScript code and attaching scripts to objects. It is straight-forward to convert exchanges to NPC nodes, choices to PC nodes, and links to link nodes. If a topic is linked by multiple choices, the first exchange in the topic is converted to an NPC node and placed as a child under the first PC node. The other links are converted to link nodes under their respective PC nodes. Generating **When** and **What** scripts for conversation nodes will be described in Section 4.1.

3.6 Summary

This chapter described the structural components of the dialogue pattern model. Topics are composed of zero or more inner exchanges and one tail exchange. An exchange corresponds to a NPC node in the Aurora conversation editor. Each exchange can have choices, which correspond to PC nodes in the Aurora conversation editor. Then, the linking of these topics was described and end dialogue targets were introduced. Next, topic groups were described as a way of sharing choices among

several topics. Finally, this chapter described how to generate these components into a Neverwinter Nights module by translating the components into nodes in the Aurora conversation editor.

Chapter 4

Dialogue Patterns

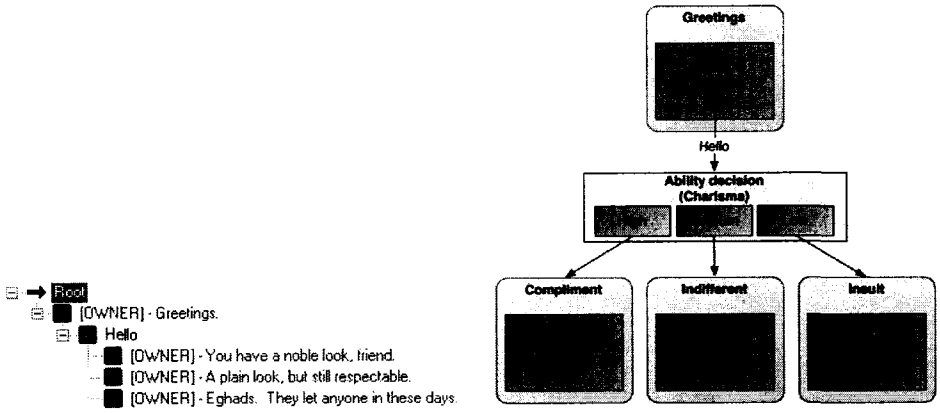
A *dialogue pattern* is the combination of structural patterns described in Section 3 integrated with decision patterns and optional choice patterns described in this chapter. A dialogue pattern can be instantiated as a sub-tree either into a conversation for an NPC or a larger dialogue pattern. The smallest dialogue pattern can be merely a single topic with one exchange and one choice. Although the author can pre-build entire dialogue patterns, they are mostly instantiated from the *disconnected bin* described in Section 5.7.

4.1 Decision Patterns

Although structural patterns allow the author to set remark text on NPC and PC remarks, they do not directly support the attachment of **When** and **What** scripts. Following the ScriptEase approach, **When** and **What** scripts are generated by patterns. Section 1.4.3 describes that for the Aurora conversation editor if an NPC node has siblings, then **When** scripts are used to select one of the siblings for display. The first **When** script that evaluates to TRUE is displayed. Essentially the NPC is making a *decision* on what remark to speak based on the game state. For example, in Figure 4.1(a), there are three NPC sibling nodes after the "Hello" PC node. The first and second sibling nodes both have **When** scripts attached that check to see if the PC's *charisma*[1] ability is above a certain value. These **When** scripts are indicated by the green "~" in the node's blue box icon. If the PC's charisma is considered *high* (above 14), the first NPC sibling will be displayed. If the PC's charisma is *normal* (between 10 and 14 inclusively), the second NPC sibling is displayed. The third NPC node has no script attached, and is displayed by *default*, i.e. when all previous sibling scripts evaluate to false. In this case, it is displayed when the PC's charisma is considered to be *low* (below 10).

In the dialogue pattern model, an NPC decision can be encapsulated as a *decision pattern*. A decision pattern allows an NPC to make a decision based on a single criterion. The decision can have two or more outcomes, and the author can choose which topics will be selected for each outcome.

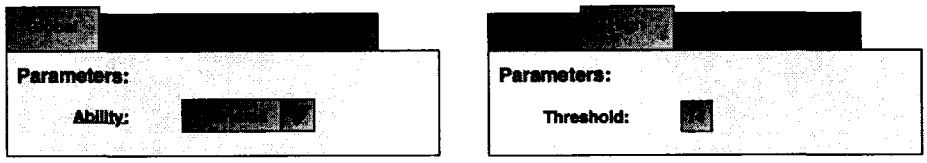
[1] Charisma determines a character's physical attractiveness and personality. It is commonly used in conversations to decide how the NPC reacts to the PC.



(a) Aurora conversation editor.

(b) Dialogue pattern model.

Figure 4.1: An *Ability* decision pattern based on the PC's charisma.



(a) Global option window pane.

(b) "High" outcome window pane.

Figure 4.2: An example of a GUI to set decision pattern options.

For example, in Figure 4.1(b), the charisma example is represented by an *Ability* decision pattern with the ability set to *charisma*. The pattern is displayed as a labeled green box, with the outcomes displayed as tabbed boxes. Each *outcome tab* has a brief label description. For example, the "High" outcome is selected if the PC's charisma is greater than 14. Similar to linking to a topic, the "Hello" choice directly links to the decision. Each outcome tab also has a direct link to a topic. The author does not see the scripts; they are generated by a decision pattern. Notice that in the Aurora conversation editor the fact that the NPC decision is based on an ability score (charisma) is also hidden with the scripts. With decision patterns the scripts are hidden but the decision intent is clearly visible. Each decision pattern instance and the outcomes inside the instance can also be uniquely labeled for easy identification. For example, the decision pattern in 4.1(b) is labeled "Charisma". The labels of the outcomes can also be changed.

4.1.1 Decision Options

Similar to encounter and behavior patterns, decision patterns have options that can be set to customize the pattern to a specific instance solution. A decision pattern can have two types of options: *global options* and *outcome options*. Global options affect the entire pattern instance and can be any type in the ScriptEase type system. For example, the *Ability* decision pattern has a single global option: the **Ability** option. The author can set it to any of the six ability statistics, such as wisdom, charisma, etc. In Figure 4.1(b), it has been set to charisma. In general, global options affect how all of the outcomes generate scripting code. For this example, all generated scripting code will retrieve the PC's charisma ability statistic.

An outcome option is specific to a single outcome. It is used to determine whether a specific outcome evaluates to TRUE or FALSE. For example, the "High" outcome in Figure 4.1(b) has a single integer option called "Threshold" which is set to 14. The author can set this option to control how much charisma the PC must have to be considered to have *high* charisma. Figure 4.2 shows what an interface could look like to set global options and outcome options. This interface is similar to the interface used by encounter and behavior patterns in ScriptEase to set their options. However, the option window has one tab for global options and individual tabs for the outcome options. In Figure 4.2(a), the author has set the global **Ability** option to "Charisma". The "High" outcome tab is highlighted in Figure 4.2(b), where the author has entered "14" for the Threshold integer option.

While most outcomes can have options, the right-most outcome requires special consideration. Since it is the *default* outcome, i.e. it is selected if the conditions for all other outcomes are not satisfied, it does not need a script. Therefore it does not need any options. This applies to all decision patterns, regardless of the number of outcomes. For example, a decision pattern with only two outcomes will have the right-most outcome as *default*. Consequently, only the first outcome will have options that the author can set.

4.1.2 Code Generation

Before a decision pattern can generate scripting code, the conversation must first be translated to Aurora's native conversation format as described in Section 3.5. For example, when the author wants to generate code for the charisma conversation in Figure 4.1(a), it is first converted to the Aurora conversation in Figure 4.1(b). The exchanges in the topics linked by the decision pattern's outcomes are converted to sibling NPC nodes. The decision pattern then generates the **When** scripts for the "You have a noble look..." (high) and "A plain look..." (medium) NPC nodes. The first **When** script is generated by using the global **Ability** option (Charisma) and the **Threshold** option for the "High" outcome (14):

```
int StartingConditional()
{
   int HIGH_OUTCOME_THRESHOLD = 14;
   if (GetAbilityScore(GetPCSpeaker(), ABILITY_CHARISMA) > HIGH_OUTCOME_THRESHOLD) {
     return TRUE;
   }
   return FALSE;
}
```

A similar script is generated for the second script attached to the "A plain look..." (medium) NPC node. Since the scripts are generated starting from the left outcome, the right-most outcome (low) is the *default* case and produces no script.

4.1.3 Sample Decision Patterns

In addition to the *Ability* decision pattern, this section introduces several other interesting decision patterns. These patterns are used to convert Aurora conversations in Chapter 1 of the official campaign into dialogue patterns. A complete set of decision patterns are presented in Appendix A. The dialogue patterns presented here are repeated in the appendix.

Basic gender Decision

The *Basic gender* decision decides on the PC's gender. Although the player can only create male and female characters, Neverwinter Nights identifies several different genders for NPCs: female, male, both, and neutral. This decision is useful to differentiate between female and male player characters. The *Basic gender* decision has "Female" and "Male" outcomes with "Male" as the *default* outcome. Similar to the *Ability* decision pattern, this pattern is designed to decide specifically on a PC characteristic, and therefore requires no additional options to function properly. A more general *Gender* decision provides outcomes for all possible genders, based on any target creature provided as a global option.

Door locked Decision

The *Door locked* decision decides on the locked status of a door. A door can either be "Locked" or "Unlocked" and the pattern has an outcome for each state with "Unlocked" as the *default* outcome.

The pattern also has a single global option **Door** so that the author can select the target door. Unlike the *Ability* and *Basic gender* decisions, the *Door locked* decision does not use PC characteristics to select an outcome.

Near by Decision

The *Near by* decision decides whether a game object is within a certain distance of another game object. The "Inside" outcome is the first outcome and is selected when the two objects are within a certain distance. The "Outside" outcome is the *default* outcome. The pattern has three global options. Both the **First Object** and **Second Object** options can be any game object. The **Distance** option is a *float* representing the distance in meters.

Progress Decision

The *Progress* decision decides on an outcome based on which remarks in the conversation have been previously *visited*. A NPC remark is considered visited if the remark is displayed in conversation. A PC remark is visited if the player selects it as a choice in the conversation. For example, a conversation can make an early decision on whether the NPC greets the PC with either "Hello stranger!" or "So we meet again!". The first remark is selected if it is the first time the PC has conversed with the NPC. The second remark is selected for all subsequent conversations. This decision selects an outcome based on whether the "Hello stranger" remark was previously visited. If it has been previously visited then the decision selects the "So we meet again!" outcome.

The *Progress* decision has two outcomes. The first outcome, labeled "Initial", has one remark option called **Goal**. Similar to other game objects in ScriptEase, the author would select the **Goal** remark from a picker. The **Goal** option specifies which remark needs to be visited in order for the "Initial" outcome to be *not* selected. For example, the author can select the the "Hello stranger!" remark as the **Goal** option. The second outcome, "Final", is the *default* outcome. This outcome is always selected after the **Goal** remark has been reached. In this example, the "Initial" outcome links to the topic with the "Hello Stranger" remark which is same as the **Goal** option. This allows the outcome to be selected once and only once, which is useful for first-time greetings in conversations.

In a second example, the author may want the PC to ask the NPC for a favour. Using a *Progress* decision, the NPC's reply can either be "Sure, I'll help." for the "Initial" outcome or "You've said enough. Goodbye." for the "Final" outcome. The "Final" outcome is selected if the PC insults the NPC in another part of the conversation by visiting the "You have a face only a mother could love." remark. This is done by setting the **Goal** option for the "Initial" outcome to this insulting remark. Now the NPC would be happy to assist the PC unless the PC decides to insult the NPC.

The adaptations described in Section 4.1.4 give the *Progress* decision extra flexibility. Each new outcome includes its own **Goal** option, allowing the author to create any number of "phases" for a single decision. Intuitively, the decision "progresses" from the first outcome to the final *default*

outcome as more goal remarks in the conversation are visited. The pattern can also be generalized by changing the **Goal** option from a remark to a *list* of remarks. In this case, if any one remark in the list is visited during conversation, then the outcome will no longer be selected.

Section 4.1.2 describes that decision patterns generate **When** scripts for NPC conversation nodes that put conditions on whether a remark appears in the conversation. However, for the *Progress* decision pattern to function, it needs to associate actions with each remark that is specified as an **Goal** option. These actions set a local variable on the NPC to indicate which remarks have been visited. The *Progress* decision pattern uses these local variables to determine which outcome to select.

Recall Decision

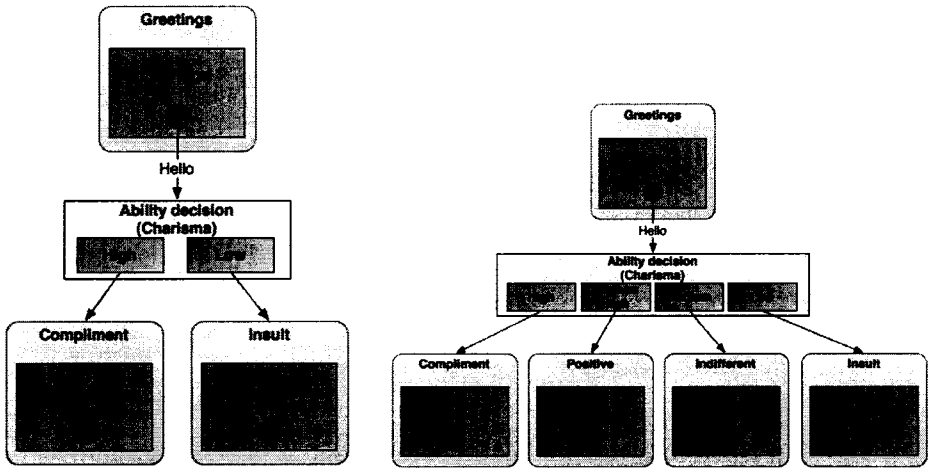
The *Recall* decision makes a decision based on a small piece of game state that was stored at a certain point in a conversation. The author can use this pattern to make a decision on information that was relevant at an arbitrary point in an arbitrary conversation. For example, the author wants the NPC to greet the PC differently depending on whether the PC lied to the NPC about having a special item earlier in the conversation. The PC has the choice to lie or tell the truth, and that choice is recorded by the decision. When the PC talks to the NPC a second time, the decision can then *recall* the recorded information to decide how the NPC will greet the PC.

The *Recall* decision has a "First" outcome and "Last" *default* outcome. The pattern has a remark global option called **Point of Interest** which represents the remark in the conversation where the decision needs to remember a piece of game state. The decision pattern remembers information in the form of strings. The "First" outcome has a **Value** string option. This option is compared against the string stored when the **Point of Interest** remark was visited. If the strings match, the outcome is selected. Otherwise the default "Last" outcome is selected. The pattern can be adapted by adding additional outcomes, where each outcome's **Value** string option is compared against the remembered string.

Similar to the *Progress* decision, the *Recall* decision requires actions to be attached to the **Point of Interest** remark. These actions store a string as a local variable on the NPC. However, the *Recall* decision differs from the *Progress* decision since the remark option is a global option rather than a outcome option. Also, the pattern decides on game state stored when the **Point of Interest** remark was visited, and not on whether the remark was visited. Consequently, the author needs to specify the piece of game state that the *Recall* pattern uses by adapting the actions that are stored on the **Point of Interest** remark. This process is described in Section 4.1.5.

4.1.4 Adaptations

Similar to other ScriptEase patterns, an author may want to further adapt a decision pattern in the context of a particular story. Besides setting options, decision patterns can be adapted by either



(a) Removing the "Medium" outcome.

(b) Adding the "Medium High" outcome.

Figure 4.3: Adapting a *Ability* decision pattern.

adding or removing outcomes. For example, the author may only want to have the NPC make a decision based on whether the PC has only *high* or *low* charisma. Since the *Ability* pattern has 3 outcomes, it is unsuitable for this situation without adaptation. Instead, the author can adapt it by removing the "Medium" outcome and controlling the threshold between *high* and *low* charisma by setting the **Threshold** option for the "High" outcome. Figure 4.3(a) shows the adapted *Ability* decision pattern.

The author can also add a new outcome to the decision pattern, before the right-most *default* outcome. In the simplest case it has the same options as the other outcomes. For example, to construct a more detailed *Ability* decision pattern, the author could insert a new outcome called "Medium High" between the "High" and "Medium" outcomes. Figure 4.3(b) shows the adapted pattern. In the more complex case, the author adapts the condition for the outcome by selecting a new condition. All of the definitions and conditions available for encounter patterns may be used in decisions patterns. The author may also need to add additional outcome options.

4.1.5 Building Decision Patterns

Section 2.3 discussed that both encounter and behavior patterns can be created by using the same operations that are used to adapt existing patterns. Similarly, decision patterns can be constructed by using the adaptations described in Section 4.1.4. A new design pattern starts with only a single *default* outcome. The author can then add new outcomes. A GUI would provide support to set the name for each outcome as well as the pattern. Next the author can add additional *global* or *outcome* options. These options are then used to set the condition for each outcome. The outcome condition is composed of ScriptEase *definition* and *condition* components, as described in Section

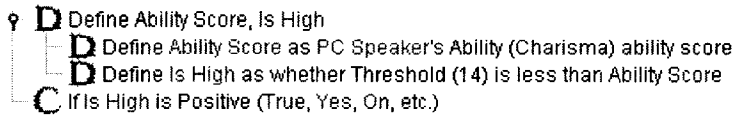


Figure 4.4: The condition for the "High" outcome in the *Ability* decision.

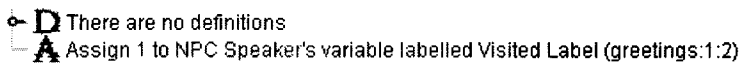


Figure 4.5: The action attached to conversation node options in the *Progress* decision.

2.3. For example, in Figure 4.4 the condition for the "High" outcome in the *Ability* pattern has 2 definitions and 1 condition. The first definition defines the PC's ability score using the global **Ability** option. The second definition defines a binary *less-than* comparison between the ability score and the outcome's **Threshold** option. Finally, the condition evaluates the comparison, returning TRUE if the comparison is TRUE.

If an option is a single remark node, then the author can specify a list of *actions* that will generate code in the remark's **What** script. If the option is a list of remarks, then the code will be generated in the **What** script for each remark in the list. *Progress* decisions use this technique to record whether the **Goal** remarks have been visited. For example, Figure 4.5 shows the action associated with the **Goal** option for the "Initial" outcome. This action sets the value of a variable on the NPC speaker object to 1. The variable's label encodes the identifier of the remark which allows the decision pattern to identify which remarks have been visited. A second example is the *Recall* decision described in Section 4.1.3. It uses an action to store a local string variable on the NPC. The author must choose the correct action depending on the game state to be stored. Although actions are the only component needed, the author could also include definitions and conditions before the actions, similar to an situation in an encounter pattern.

4.1.6 Composing Decision Patterns

Decision patterns are designed to make a decision along a single criterion, such as one of the PC's ability scores, whether a door is locked, or progression through a conversation. However, many complex conversations, including conversations in the official campaign, make decisions involving multiple criteria. For example, in Chapter 1 of the official campaign, the NPC named Emernick has three possible opening remarks when the PC initiates conversation. These three outcomes depend on two criteria: a) is the saferoom door locked and b) is Emernick close to the saferoom waypoint object (i.e. inside the saferoom). Outcome 1 occurs if the saferoom door is locked, since it is assumed Emernick is safely inside and is willing to answer the PC's questions. Outcome 2 occurs if the saferoom door is unlocked *AND* Emernick is close to the saferoom waypoint object (i.e. inside the saferoom). In this case Emernick will instruct the PC to lock the door with the lever. Finally,

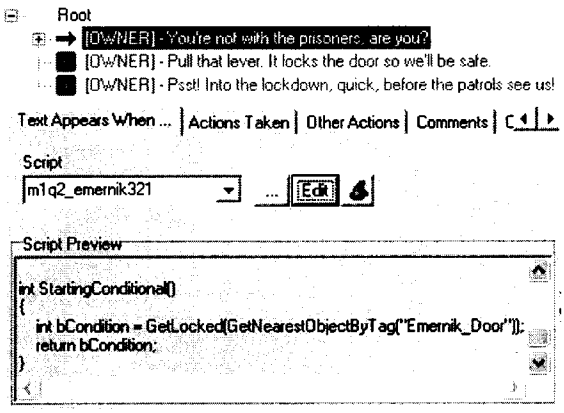


Figure 4.6: The first decision of the Emernick NPC in the Aurora conversation editor.

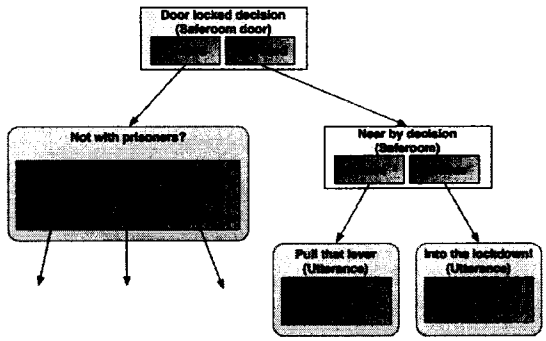


Figure 4.7: The first decision of the Emernick NPC is composed with 2 decision patterns.

outcome 3 occurs if the saferoom door is unlocked *AND* Emernick is far away from the saferoom waypoint (i.e. outside the room). In this case he will instruct the PC to run with him to the saferoom. The actual Emernick conversation is more complex, but the extra decision have been remove for exposition clarity.

In the Aurora conversation editor, these three outcomes are sibling NPC nodes as shown in Figure 4.6. Outcomes 1 and 2 have **When** scripts attached with outcome 3 as the *default* outcome. The first script simply evaluates to *TRUE* if the saferoom door is locked. The second script assumes the saferoom door is unlocked – otherwise the first would be already selected – and returns *TRUE* if Emernick is close (i.e. within 3 meters) to the saferoom waypoint.

If the author wanted to convert this conversation to the dialogue pattern model, it is unlikely that there exists a pre-built decision pattern that handles this specific combined decision. With some ScriptEase experience, the author could build a new decision pattern for this specific example by creating each outcome and setting the conditions manually. Unfortunately this is a tedious process and increases the likelihood of a cluttered pattern catalog with specialized patterns that are used infrequently.

Instead, the author can create a composite decision by linking two or more existing decision

patterns together. For example, to create Emernick's three decision outcomes, the author uses two more general decision patterns. The *Door locked* decision has two outcomes based on whether a door (e.g. Saferoom door) is locked or unlocked. The *Near by* decision also has two outcomes based on whether an object (e.g. Emernick) is within a certain distance (e.g. 3 meters) of another object (e.g. Saferoom waypoint). Both are general patterns that can be reused for a variety of decisions. By linking the "Unlocked" outcome in the *Door locked* decision to the *Near by* decision, as shown in Figure 4.7, the author can convert Emernick's three outcome decisions to the dialogue pattern model.

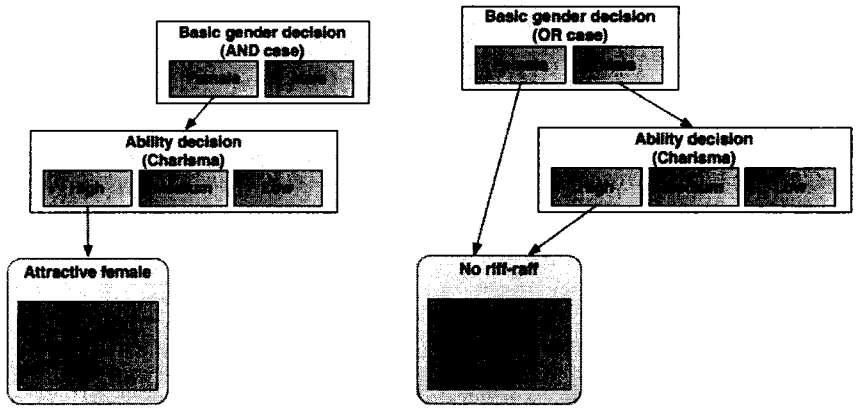
It is possible to convert any outcome that depends on several conditions connected with boolean operators. The Emernick example illustrates an *AND* operator. As a canonical example of the AND case, consider an outcome that is selected only if the PC is female *AND* the PC has *high* charisma. By itself, the *PC is female* condition is an outcome in the *Basic gender* decision. As seen previously, the *PC has high charisma* condition is an outcome in the *Ability* decision adapted to charisma. By linking the "Female" outcome to the *Ability* decision and the "High" outcome to the topic, as shown in Figure 4.8(a), the original outcome can be achieved. Alternatively, the "High" outcome can be linked to the *Basic gender* decision and the "Female" outcome to the topic for the same result.

As a canonical example of the *OR* case, consider an outcome that is selected if the PC is female *OR* the PC has *high* charisma. Again the *Basic gender* and *Ability* decision patterns are linked together, but this time the "Male" outcome is linked to the *Ability* pattern. Also, both the "Female" and the "High" outcomes now link to the topic, as shown in Figure 4.8(b). The two direct links indicate that either outcome will select the topic to be displayed. In the *AND* case, there is only a single path that goes through both decisions since the topic requires an outcome from both decisions. In the *OR* case, there are two paths to the topic. One path which passes through a single decision and one that passes through both decisions.

Since all complex **When** scripts are composed of basic conditions connected with *AND* and *OR* boolean operators, any **When** script attached to an NPC node can be represented with one or more single-criterion decision patterns.

4.1.7 Degenerate Decision Patterns

Similar to a choice, an outcome of a decision pattern links to a topic. This topic may be the root of an entire sub-tree. However, there are many cases where the author only wants a decision to affect a single exchange where each outcome of the decision shares the same choices of the exchange. For example, the nurse NPC in Chapter 1 of the official campaign has a different greeting (remark) depending on the PC's charisma ability score. Regardless of the greeting, the PC has the same set of choices to respond to the nurse. Figure 4.9(a) shows this piece of conversation in the dialogue pattern model. An *Ability* decision has its three outcomes pointing to different topics in a topic group. The topic group has three topics that share the same set of choices with each topic having a



(a) Topic is displayed if the PC is female *AND* has high charisma.

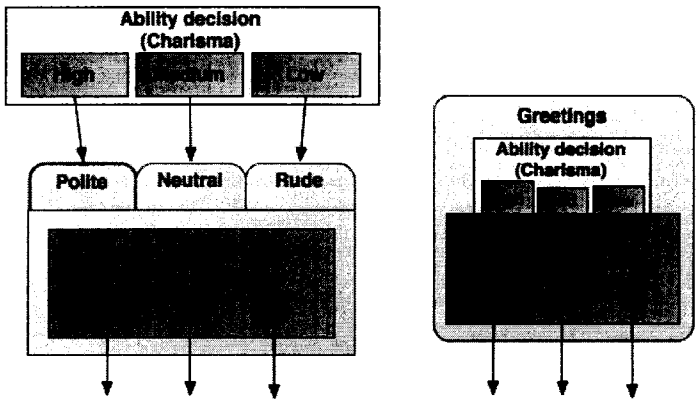
(b) Topic is displayed if the PC is female *OR* has high charisma.

Figure 4.8: Using combinations of decision pattern to simulate logical operators.

single exchange. The author can set the exchange remark text for each topic based on the outcome linking to the topic.

Although using a topic group for a decision on a single NPC remark is convenient, this usage of decision patterns occurs frequently enough that a more compact representation would be useful. For example, the nurse conversation uses a decision pattern in five different places to decide on only a single NPC remark. Instead, the decision pattern can be instantiated as a *degenerate decision pattern*. In this representation (Figure 4.9(b)), the decision pattern is attached directly to an exchange inside a topic and indicates that there exists one exchange for each outcome. The decision's outcome tabs function similar to the tabs in a topic group by allowing the author to change the visible exchange by clicking on an outcome tab. For example, in Figure 4.9(b), the *Ability* decision is now a degenerate decision pattern connected directly to the exchange. There are still three exchanges with one for each outcome, but the topic group is no longer necessary. This is a slightly more compact representation that clearly indicates the author's intent. If a decision is not a degenerate decision pattern, it is called a *normal decision pattern*.

In contrast, the same structure is much more complicated in the Aurora conversation editor. The author must first create an NPC node for each outcome. Then, the author constructs the remaining conversation under the first NPC sibling node. Finally, the author creates a link node under each of the remaining sibling nodes. Figure 4.10 shows the same section of the nurse's conversation as it would appear in the Aurora conversation editor. This decision requires three NPC nodes with three PC nodes under the first NPC node and a total of six link nodes under the other two NPC nodes. When these single-remark decisions are used frequently, these links can greatly inflate the size of the Aurora conversation tree. For example, there is one case described in Chapter 6 where the Aurora



(a) A normal decision pattern with outcomes linking to topics in a topic group.

(b) The topic group is replaced with a degenerate decision attached to the exchange.

Figure 4.9: Simplifying decisions that affect only a single NPC remark.

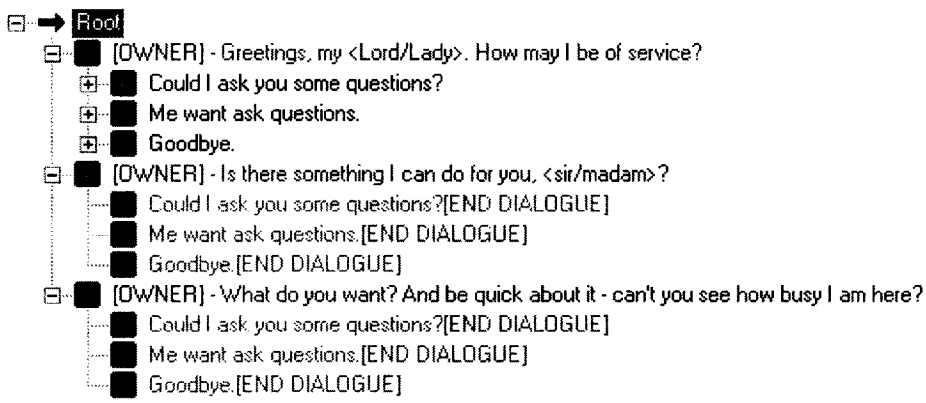
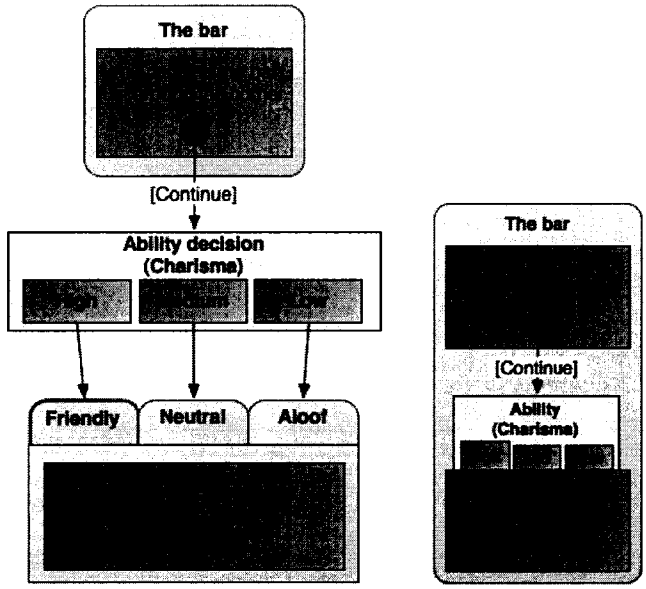


Figure 4.10: The nurse's greeting decision in the Aurora conversation editor.

Nurse conversation has a 189 total conversation nodes. Of these 189 nodes, 125 nodes (66%) are link nodes. Two thirds of the conversation redirects the author back to nodes in the other one third of the conversation.

There is a second advantage to using degenerate decision that occurs when using large topics. For example, consider a topic that has an exchange length of two (i.e. two exchanges, one linking to the other). If the author wanted a regular decision pattern to decide on the remark text in the second exchange, the topic would have to be split into two separate topics each with a single exchange, as shown in 4.11(a). The author would then link the first topic to the decision pattern, and the outcomes of the decision pattern to the second topic which is now a topic group. In addition to the complexity required when degenerate decision are not available, this structure may lose the intent



(a) A normal decision pattern requires the topic to be split into two topics.

(b) A degenerate decision allows the topic to remain intact.

Figure 4.11: A decision pattern that affects only a single exchange inside a topic with two exchanges.

of the conversation. In the example of Figure 4.11, the author intended both exchanges to be part of the same original conversational topic ("The Bar") with the second remark customized based on game state. If degenerate decisions are used (Figure 4.11(b)) both exchanges can remain inside the same topic, preserving the author's original intent.

4.2 Optional Choice Patterns

When constructing choices, an author may want certain choices to be available only when certain conditions are met. For example, the PC might have a insightful response to the NPC that can reveal extra information. However, this choice is only available if the PC has a high *wisdom* ability score. Section 1.4.3 describes how the Aurora conversation editor enforces this condition. The author attaches a **When** script to the PC node. The script evaluates to TRUE if the PC's wisdom is high, and FALSE otherwise.

In dialogue patterns, the author can make a choice optional by instantiating an optional choice pattern. Similar to other ScriptEase patterns, an optional choice pattern can have options. These options are used as parameters for the pattern's condition. Similar to conditions for outcomes in decision patterns, the optional choice condition consists of ScriptEase definitions and a single ScriptEase condition, and generates code for a **When** script that is attached to the choice. For example, Figure

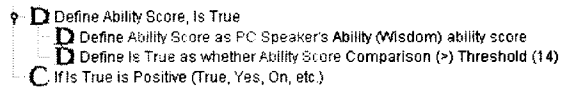


Figure 4.12: The condition for an *Ability* optional choice pattern.

4.12 shows the condition for an *Ability* optional choice pattern that is displayed only if the PC has high wisdom. The pattern has an **Ability** option set to *wisdom*, a **Threshold** option set to 14, and a **Comparison** option set to $>$ (greater than). The **Comparison** option allows the author to choose how the ability score is compared against the threshold with $<$, $\leq$, $=$, $\geq$, or $>$ boolean comparison operators. The condition's first definition defines the PC's ability using the **Ability** option (wisdom). The second definition defines a binary $>$ (greater than) comparison between the ability score and the **Threshold** option (14). Finally, the condition evaluates the comparison, returning TRUE if the comparison is TRUE.

Several optional choice patterns can be instantiated on the same choice. In this case, the conditions for all the attached optional choices patterns must be true before the choice becomes available. For example, if a second *Ability* optional choice pattern (*intelligence* $>$ 9) is attached to the same choice as the *Ability* pattern in Figure 4.12, then the PC must have both a wisdom ability score of more than 14 *AND* an intelligence ability score of more than 9.

Other useful optional choice patterns include the *Has item* and *Quest point* patterns. The *Has item* pattern makes a choice available only if a specific item in the PC's inventory. The author can set the item with the pattern's **Item** option. The *Quest point* pattern makes a choice available only if a certain point is reached in a quest. This is useful to make sections of a conversation available only if the PC is at a certain point in a quest. For example, an NPC might be willing to give the PC some gold, but only if they completed an *Retrieve an item* quest first. These optional choice patterns are described in more detail in Appendix A.

The Aurora conversation editor has two important built-in scripts. The "normal int" script returns TRUE if the PC has a normal or greater intelligence ability score (i.e. 9 or more). The "low int" script returns TRUE if the PC has a low intelligence ability score. In all conversations in the official campaign, the PC's choices depend on the the PC's intelligence. If the PC has low intelligence, most choices will have "dumbed down" remark text to reflect the PC's lack of sophisticated speech. To do this, the author creates two choices, one for the "normal" choice with the "normal int" script attached, and the other for the "dumbed down" choice with the "low int" script attached. Both scripts are designed to be mutually exclusive, i.e. for any condition one script evaluates to FALSE and the other to TRUE. Consequently, only one of the two choices will be displayed for any given conversation.

This frequent pattern can be created easily using two *Ability* optional choice patterns – one attached to each choice – with the **Ability** option set to *intelligence*. The pattern for the "normal" choice has a **Threshold** of 9 with a $\geq$ **Comparison**. The pattern for the "dumbed down" choice has

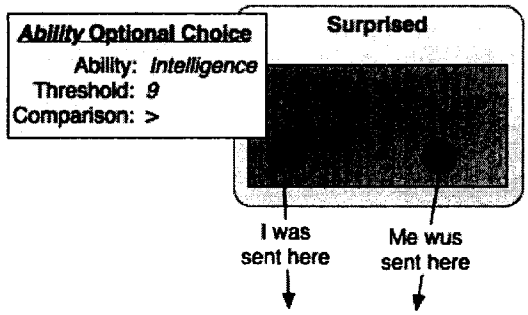


Figure 4.13: An exchange with 4 *Ability* optional choice patterns. The simulated pop-up window shows details of the first optional choice pattern.

a **Threshold** of 9 with a < **Comparison**. For example, in Figure 4.13 the exchange has 4 choices. They are marked as optional choices with a "?" symbol. With the current representation of the model, optional choice patterns cannot be displayed explicitly in diagrams since choices are too small to contain details such as the pattern's name. Instead, the author must click on the optional choice to view the pattern. Similar to **When** scripts in the Aurora conversation editor, the "?" symbol only indicates that a pattern exists, but not the name of the pattern. In Figure 4.13 a mouse-over pop-up window has been simulated that shows information about the *Ability* optional choice pattern attached to the first choice. The first and third choice are considered "normal" choices, and the second and fourth choices are "dumbed down" choices.

4.2.1 Choice Groups

For both the Aurora conversation editor and dialogue patterns, the widespread use of intelligence-based choices creates extra complexity. For example, Figure 4.14 shows a portion of Emernick's conversation in Chapter 1 of the official campaign. There are 10 PC nodes, however the first five nodes require "high int" **When** scripts. The second five nodes are low intelligence versions of the first five remarks. Since each intelligence pair links to the same NPC node, the "dumbed down" PC nodes have link nodes to the NPC node. This adds a total of 10 extra lines or nodes to the conversation (five for low intelligence PC nodes and five for link nodes). In addition to the extra lines, the semantic relationship between the pairs of remarks is not explicit. Figure 4.15 shows the same portion of conversation as a dialogue pattern. Although the direct links eliminate the need for link targets, 10 choices are still required and there is still no explicit association between intelligence pairs.

Since the intelligence scripts are used pervasively in all conversations in the official campaign, the overhead of these conversations is significant. Although converting these conversation to dialogue patterns removes some complexity, the author still needs to duplicate each choice using normal and low intelligence variants, and then link them to topics. However, the majority of these

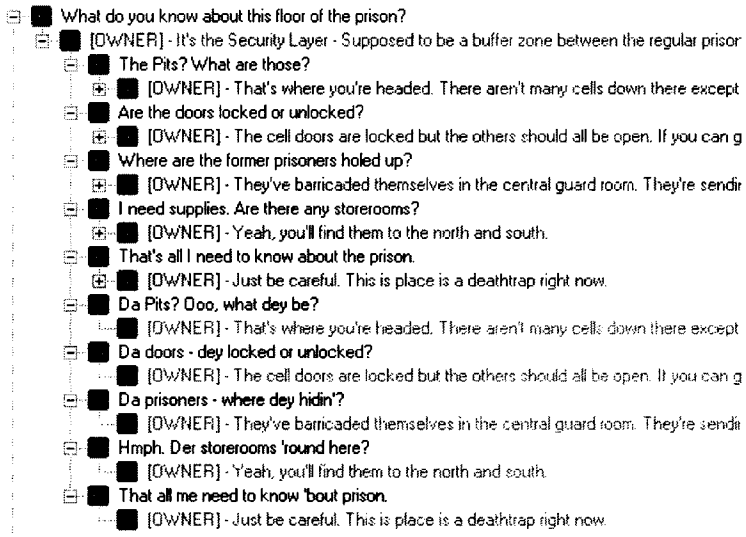


Figure 4.14: Portion of Emernick's conversation with 10 PC nodes including five normal and five low intelligence variants.

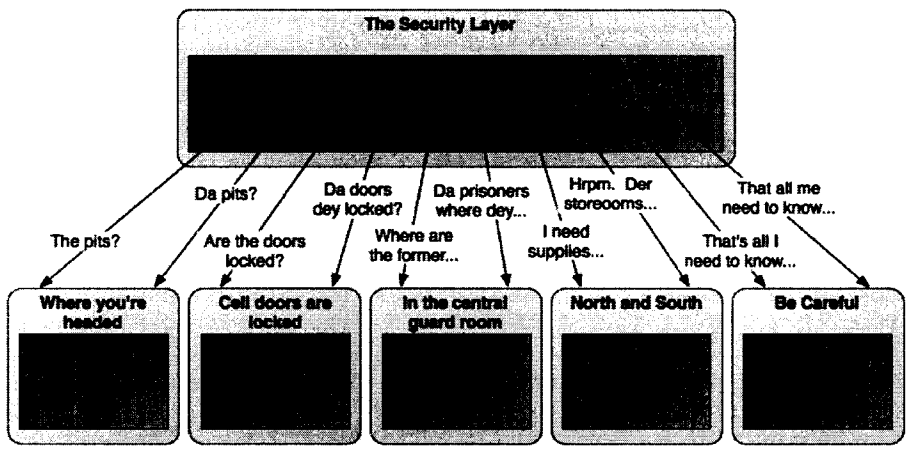


Figure 4.15: Exchange with 10 choices including five normal and five low intelligence variants.

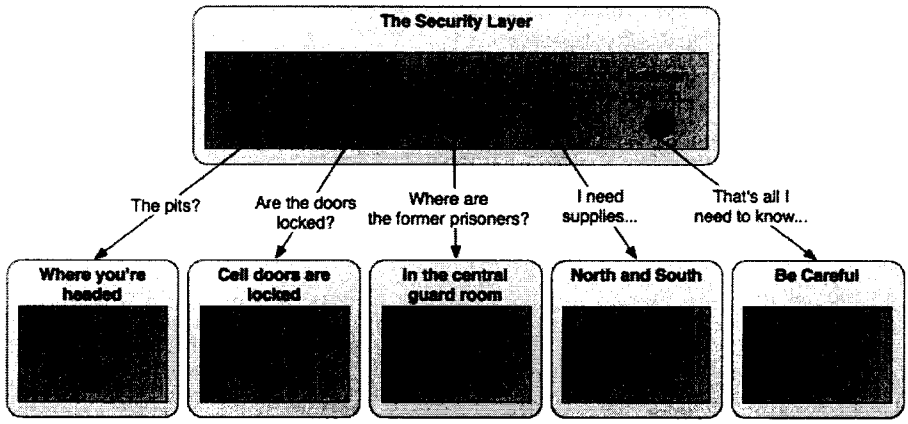


Figure 4.16: A simplified exchange with 5 choice groups each with a normal and low intelligence PC remark.

normal/low intelligence pairs always link to the same topic. For example, in all of the conversations described in the case study in Chapter 6, there are 136 total intelligence choice pairs. Of these 136 pairs, 112 pairs link to the same topic. In this common case, the author only wants to customize choice text without affecting the structure of the conversation. This case can be visually simplified by collapsing each normal/low intelligence pair into a *choice group* that explicitly shows related choices. For example, Figure 4.16 shows the piece of the Emernick conversation with choice groups. Each of the five pairs are represented by a single choice group which looks like two choice singletons stacked together. Only the remark text for the high intelligence choices are visible on the arcs with the understanding that although all nodes in the group have different text, they have same *intent* or meaning. A GUI can allow the author to view or edit individual choices in the group in a separate window. The exchange now properly represents the author's original intent. A choice that is not in a group is now called a *choice singleton*. Both a choice singleton and a choice group are considered to be *choices*.

Although the author could use a choice group with any number of choices, analysis of conversations in the official campaign suggest groups are currently only useful for normal/low intelligence pairs. All other choice singletons either are not optional, or do not share the same link with other choice singletons in the same exchange. Therefore, it is convenient to make a single choice group pattern called the *Intelligence* choice group pattern. This pattern can be used to create a choice group of two choice singletons with an *Ability* optional choice pattern on each singleton, with the **Ability** option of these patterns set to *intelligence*. Additionally, this pattern can also convert an existing choice singleton into a choice group.

Similar to a choice singleton, the author may want a choice group to be available only when certain conditions are met. If the conditions are not met, none of the PC remarks in the choice group are available to the PC. If the conditions are met, the group functions normally by making available

only one of the PC remarks. For example, an *Intelligence* choice group could have an attached *Ability* optional choice pattern set to be available only if the PC has high *wisdom*. If the PC has high wisdom and normal or better intelligence, the "normal" remark in the group is available. If the PC has high wisdom and low intelligence, the "dumbed down" remark is available. If the PC does not high wisdom, neither remark in the group is available.

4.3 Summary

This chapter has presented two types of patterns that can be integrated with structural patterns to form dialogue patterns. A decision pattern decides on what the NPC will say at a certain point in the conversation. This decision is made by evaluating conditions using game state and consequently selecting a particular path of conversation. An optional choice pattern evaluates a condition to decide on whether a specific choice is available at a certain point in the conversation. If the condition is not satisfied, the choice is unavailable. Finally, choice groups were introduced to preserve the author's intent of providing two different sets of PC remarks based on the PC's intelligence. Choices that were not groups were called *choice singletons*.

Dialogue patterns are not used to generate **What** scripts on remarks that are used to execute actions, such as rewarding the PC experience points or moving a character. These *What* patterns are already handled by encounter patterns. Specifically, a *Conversation What* encounter pattern is used to execute actions when a remark is selected or displayed. The pattern's first option is a *remark* and the author uses a picker to select the remark from a conversation. These encounter patterns typically contain actions that do not affect a conversation or its flow and consequently are not dialogue patterns. However, a GUI could mark remarks that are a part of encounter patterns and provide an operation to allow the author to view the encounter pattern.

Chapter 5

Pattern Operations

This chapter describes the set of operations needed to build, change, and delete components of dialogue patterns. By describing the operations required to build a conversation using dialogue patterns, it is possible to compare the operational complexity of creating a conversation using the Aurora conversation editor and the dialogue pattern model. A complexity metric that uses pattern operations is described in Chapter 6.

5.1 Topics

A topic is created with the **Add Topic** operation. The author must first select a *precursor* that will link to the new topic. The precursor for a topic is either a choice or decision outcome. For a new conversation, the precursor of the first topic is the *root* choice. There is always one root choice for each conversation and it does not appear in an actual conversation. The topic is created with a single exchange containing one choice. A direct link is also created from the precursor to the new topic. A GUI could have extra **Add Topic** operation variants that add a different number of exchanges and choices to the new topic. If the precursor already links to a target (i.e topic or decision pattern), the new topic is inserted between the precursor and target. The choice in the new topic links to the original target. For example, in Figure 5.1 the author creates a new topic by selecting the first choice in the "Go Away!" topic and performing the **Add Topic** operation. The new topic is inserted in front of the "Sick" topic.

A topic can also be removed with the **Remove Topic** operation. The author must first select the topic to be removed. In addition to removing the topic, the operation will remove all disconnected sub-topics. Section 5.7 describes disconnection in more detail. This operation leaves the precursor of the removed topic unconnected to any topic. For example, in Figure 5.2 the author uses the **Remove Topic** operation to remove the sub-tree rooted at the "Sick" topic.

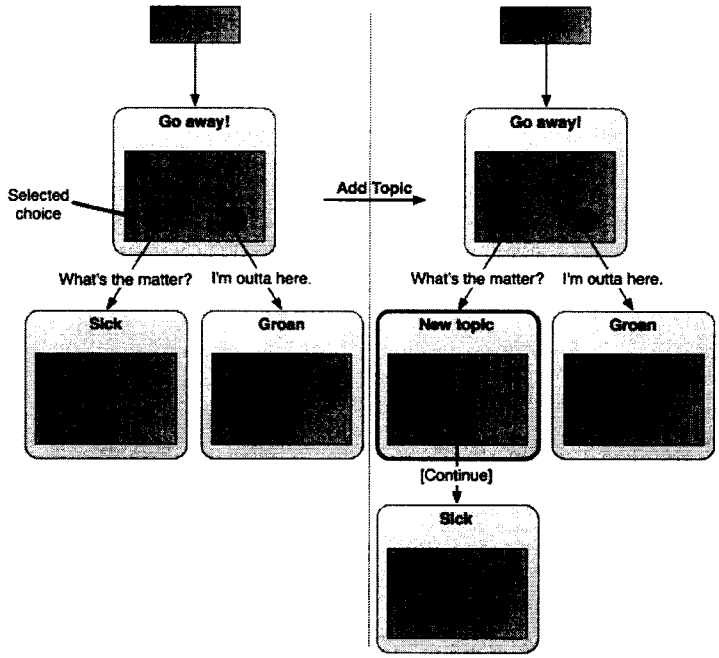


Figure 5.1: Adding a new topic to a conversation.

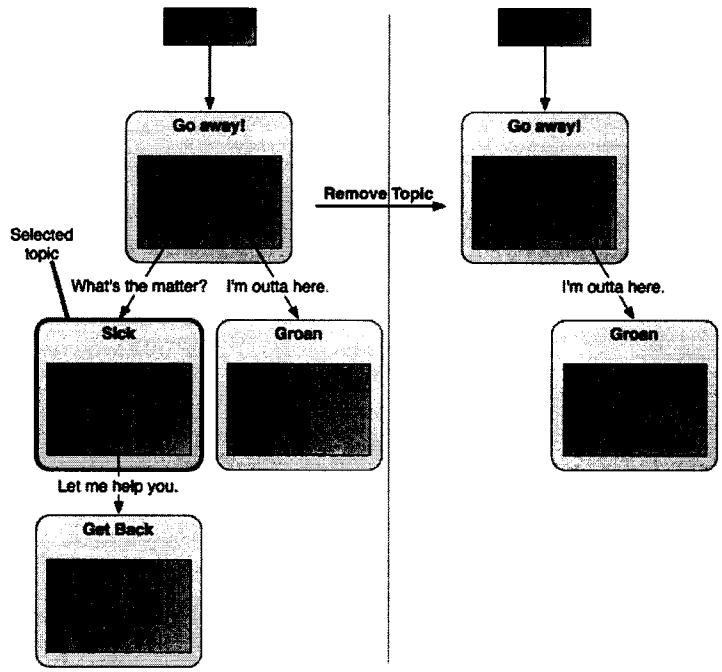


Figure 5.2: Removing a selected topic from a conversation.

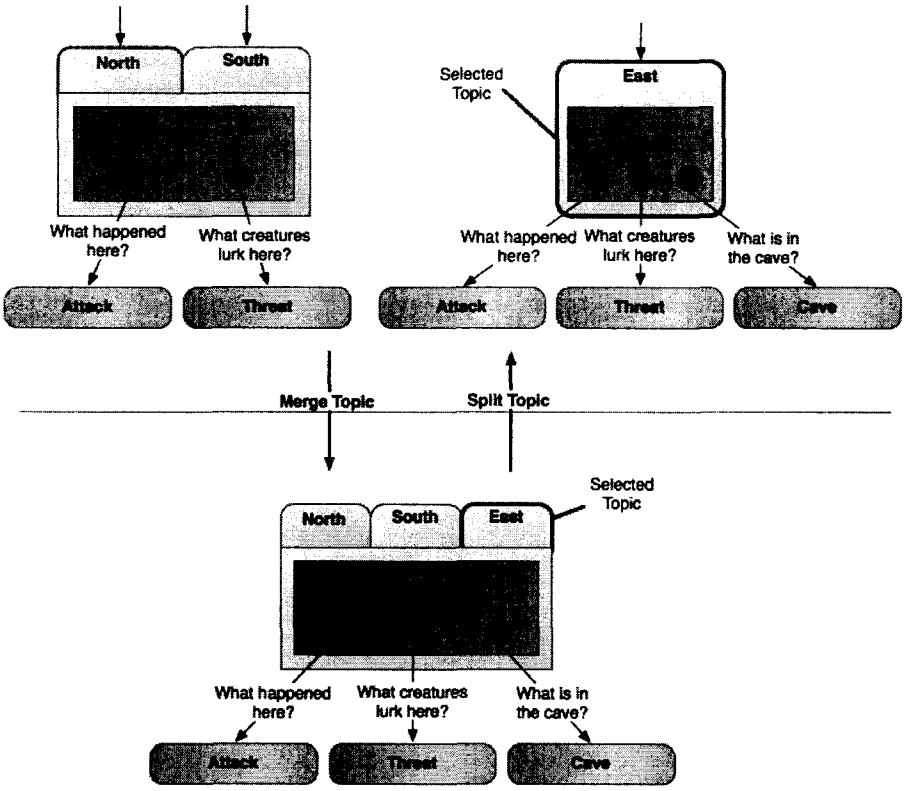


Figure 5.3: Merging a topic with a topic group. This operation can be reversed by splitting the topic from the topic group.

5.2 Topic Groups

The author can add a topic to a topic group by using the **Merge Topic** operation. The author must first select a topic group and a topic that is not in a group. The selected topic is relocated to the topic group's position and the choices in the tail exchange of the selected topic are added to the choices in the topic group. The author can then remove any unnecessary choices. For example, in Figure 5.3 the author merges the "East" topic into the topic group. The topic's three choices are added to the right of the topic group's existing choices. Since two of the three choices are duplicates of the two choices already in the topic group, they are removed. The process of recognizing duplicates can be automated by the GUI and can include or exclude author confirmation.

To create a new topic group, the author can select two topics that are not in topic groups and use the **Merge Topic** operation. The second topic will moved to the first, creating a topic group. Additionally, the author can add new topics to a topic group by selecting the topic group and using the **Add Topic** operation described in Section 5.1. The new topic is immediately attached to the topic group but remains disconnected since the author has not yet linked to the new topic from an existing precursor.

Finally, the author can remove a topic from a topic group with either the **Remove Topic** or **Split Topic** operations. The **Remove Topic** operation removes the topic from the conversation entirely. Any choices specific to the removed topic that are not shared by the remaining topics in the group are also removed. The **Split Topic** operation separates the topic from the group into a stand-alone topic. The separated topic is connected by a direct link to one of its precursors. Furthermore, all choices in the topic group that were shared by the separated topic are duplicated and added to the separated topic. Similar to the **Remove Topic** operation, any choices specific to the split topic that are not shared by other topics are removed from the topic group. For example, in Figure 5.3 the author splits the "East" topic from the topic group creating a new topic in the conversation. The new topic has the same choices as the topic group. However, since only the "East" topic has the "What is in the cave?" choice, the choice is removed from the topic group.

5.3 Exchanges

The majority of exchanges will be created implicitly when a new topic is created. However, the author can use an **Insert Exchange** operation to create a new exchanges inside an existing topic. The author must first select an existing exchange in the topic. The operation creates the new exchange before the existing exchange. The new exchange contains a choice singleton which is directly linked to the selected exchange. If the new exchange is inserted between two existing exchanges, the direct link of the first exchange is redirected to the new exchange. For example, in Figure 5.4, the author inserts a new exchange in between two exchanges in the "You're in town" topic.

Recall that the tail exchange is the only exchange in a topic that can have more than one choice. The author can use the **Append Exchange** operation to add an exchange after the selected exchange unless the selected exchange is a tail exchange with more than choice. A direct link is created from the choice in the selected exchange to the new exchange. For example, in Figure 5.5 the author is extending the topic by adding a third exchange between the two existing exchanges. This operation is disallowed with tail exchanges that have two or choices.

The author can change the move an selected exchange inside a topic using the **Move Exchange Up** and **Move Exchange Down** operators. The **Move Exchange Up** operation moves the exchange up by swapping the positions of the selected exchange and the exchange above. This operation is unavailable for the first exchange in a topic. Similarly the **Move Exchange Down** operation moves the exchange down by swapping the positions of the selected exchange and the exchange below. This operation is unavailable for the tail exchange. The direct links are adjusted to maintain consistency. For example, in Figure 5.6, the author moves the "Fine day..." exchange up to the first position in the topic. A topic's tail exchange can not be a target of this operation if it has more than one choice.

Finally, the author can remove an exchange using the **Remove Exchange** operation. However, the tail exchange cannot be removed. The author must remove the entire topic instead.

The author can set the NPC remark text inside an exchange by using the **Set Remark Text.**

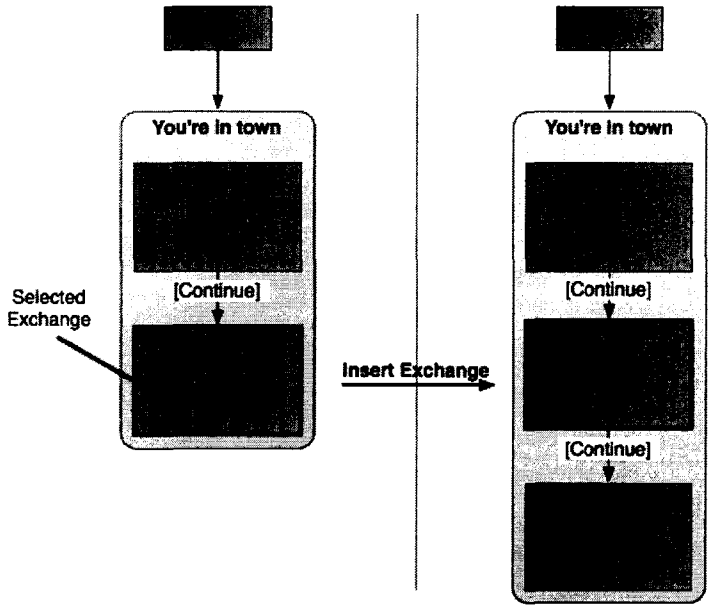


Figure 5.4: Inserting a third exchange in a topic.

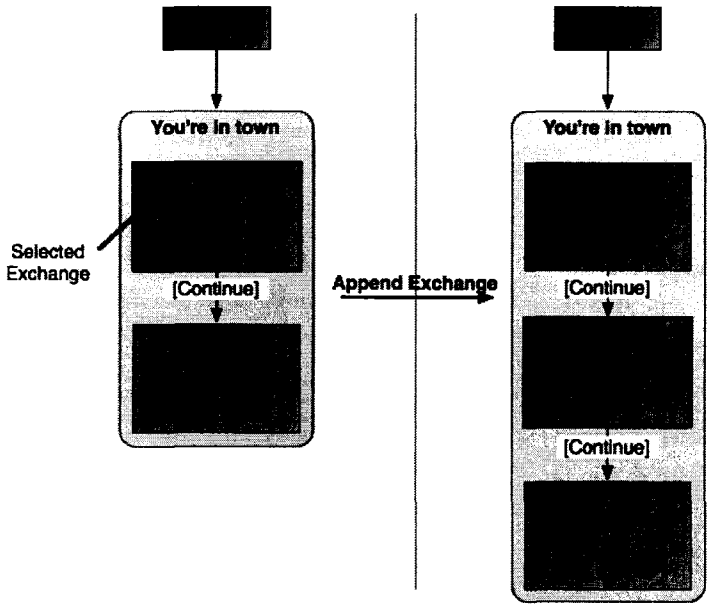


Figure 5.5: Appending a third exchange to the end of a topic.

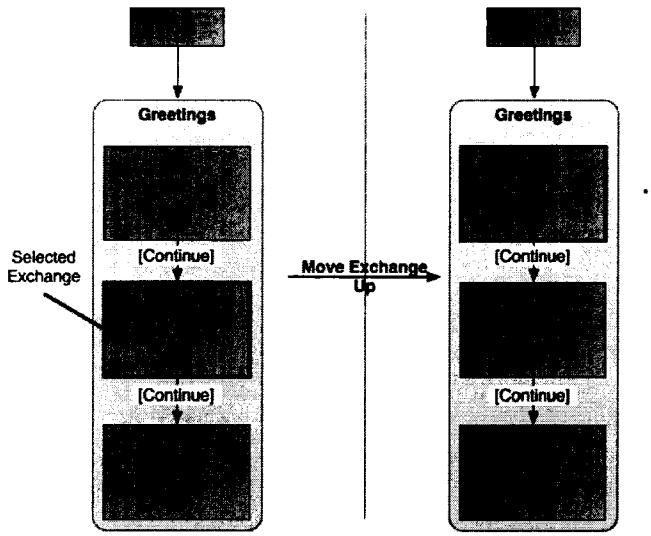


Figure 5.6: Moving two exchanges in a topic.

5.4 Choices

An author can add a new choice singleton to an exchange by using the **Add Choice Singleton** operation. Since inner exchanges can only have one choice, this operation be only be used inside the tail exchange of the topic. The author must first select an existing choice in the exchange. The operation will add a new choice singleton after the selected choice. For example, in Figure 5.7 the author adds a new choice singleton after the first choice in the exchange. Additionally, a GUI could have a **Insert Choice Singleton** that would insert a new choice singleton before the selected choice.

Section 4.2.1 describes the *Intelligence* choice group. This choice group can be created with either of two operations. The **Add Choice Group** operation functions similar to the **Add Choice Singleton** operation by adding a choice group after the selected choice. The **Convert Singleton to Group** operation converts the selected choice singleton into a choice group. This operation is only available if a choice singleton is selected. The properties of the selected choice are preserved since the choice becomes the first choice in the group. For both the **Add Choice Group** and **Convert Singleton to Group** operations, the GUI must provide a mechanism to select the type of choice group to be added. At the time of writing of this dissertation, the *Intelligence* choice group is the only type of choice group available. However, it is possible for authors to create more types of choice groups. The *Intelligence* choice group group is the only one used in Neverwinter Nights.

The author can use the **Convert Group to Singleton** to replace a choice group with a choice singleton. The GUI must provide a mechanism to allow the author to select which choice in the group will be used as the choice singleton. All other choices in the group are removed.

Similar to moving an exchange inside a topic, the author can move a choice left or right using the **Move Choice Left** and **Move Choice Right** operations. The author must first select a choice

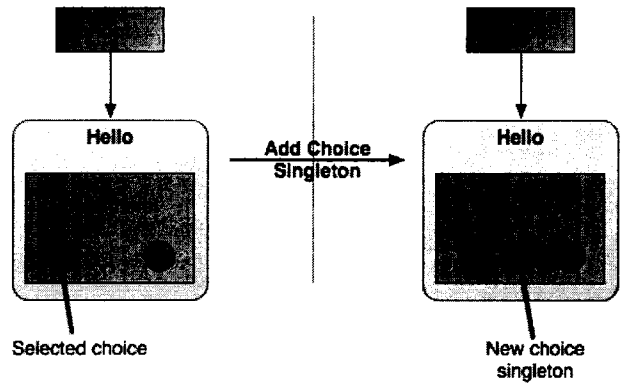


Figure 5.7: Adding a choice to an exchange.

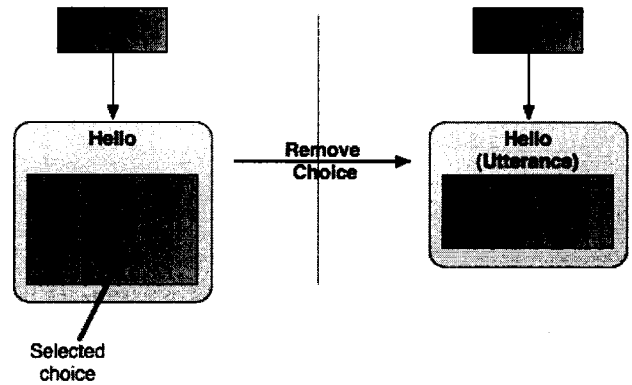


Figure 5.8: Removing a choice to create an utterance.

in a tail exchange. The **Move Choice Left** operation is only available if the selected choice has an adjacent choice to the left. Similarly the **Move Choice Right** is only available if the selected choice has an adjacent choice to right.

The author can remove a choice using the **Remove Choice** operation. The author first selects the choice to be removed. If an exchange only has a single choice, the choice cannot be removed. The only exception is if a topic has a single exchange with one choice and the topic's ancestors consist only of decision patterns and the root node. The choice can then be removed to convert the exchange into an *utterance*. For example, in Figure 5.8 the author removes the single choice to create an utterance. Utterances are described in Section 3.2. The author can convert an utterance topic back into a normal topic by selecting the exchange and using the **Add Choice** operation.

When the author removes a choice that links to a topic, the link is also removed. If the topic is not linked by another precursor, it is considered disconnected. Section 5.7 describes disconnected topics in detail.

The author can set the PC remark text inside a choice by using the **Set Remark Text**.

5.5 Linking

The author can link a precursor (choice or decision outcome) to a target (topic or decision pattern) by using the **Link** operation. The author must first select a precursor and a target. If the precursor already links to another target, the link is redirected to the selected target. This redirection can possibly disconnect a sub-tree of topics and decision patterns from the conversation. Section 5.7 describes this case in more detail.

5.6 Dialogue Patterns

Similar to adding a topic, the author can instantiate a dialogue pattern by using the **Add Dialogue Pattern** operation. The author first selects a precursor which acts as the root node of the dialogue pattern. If the precursor already has a link to another target, the target is appended to the *end* of the instantiated dialogue pattern. The *primary precursor* of the dialogue pattern is defined as the first unlinked-linked precursor in a depth-first traversal of the dialogue pattern. For example, in Figure 5.9 the author instantiates a dialogue pattern with 3 topics after the first choice in the "Faire" topic. The "South" topic is now linked from primary precursor, which is the choice singleton in the "Topic 2" topic.

Once a dialogue pattern is instantiated into a conversation, the pattern boundary disappears and the author is free to adapt the components as needed. Consequently, it is not possible for the author to remove a dialogue pattern after instantiation. The components must be removed separately.

5.7 Deletion

Whenever a component is removed, it is possible for other components to become disconnected from the conversation tree. A topic or decision pattern is *disconnected* if it is not linked by any precursor. For example, in Figure 5.10 the "Groan" topic is disconnected from the conversation. A target that is only linked by a secondary link, such as a topic in a topic group, is still considered connected to the conversation. A topic group is *disconnected* if all topics in the group are disconnected.

If a topic or topic group becomes disconnected then it is removed from the conversation tree. One exception is if a disconnected topic is part of a topic group that is still connected to the conversation. In this case, the topic remains part of the topic group, however the GUI may warn the author that the topic is not part of the conversation. For example, in Figure 5.11, the "East" topic in the topic group is disconnected and will not appear when the conversation is generated into the module. This assumes that there is no precursor anywhere in the conversation that connects to the "East" topic. The "South" topic is *not* disconnected since there is a secondary link that connects to it in the figure. The author can connect the topic by linking it with a precursor.

Formally, a topic is considered disconnected if and only if the *root* node is not an ancestor of the topic. For example, although the "Threat" and "Didn't hear" topics in Figure 5.12(a) are

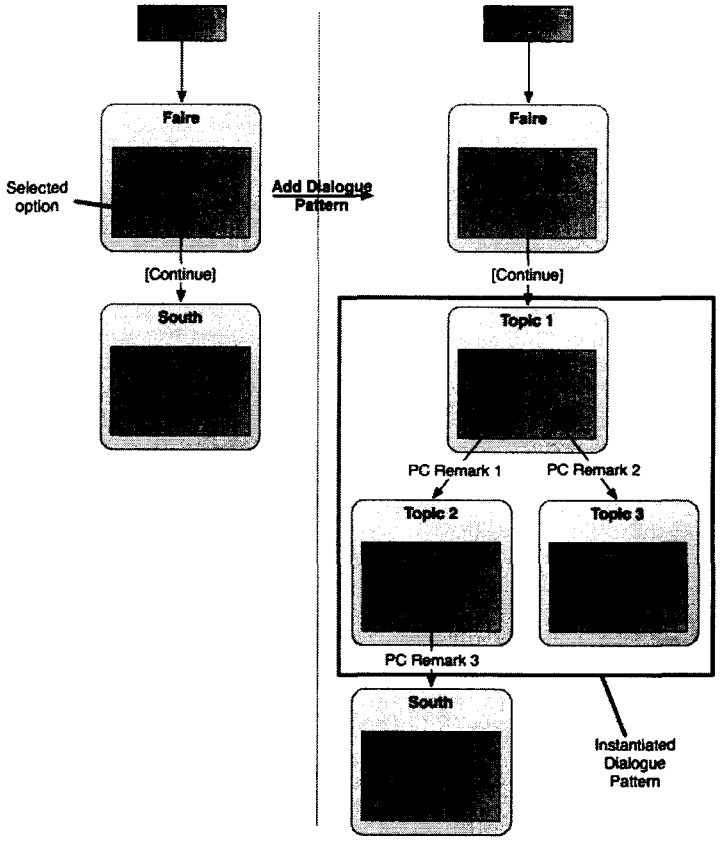


Figure 5.9: Instantiating a dialogue pattern into a conversation.

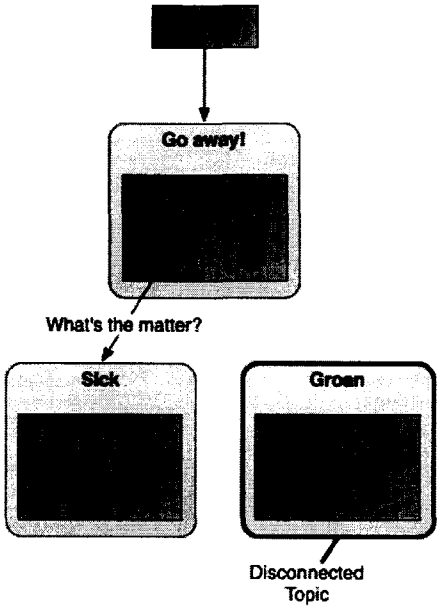


Figure 5.10: A topic disconnected from the conversation.

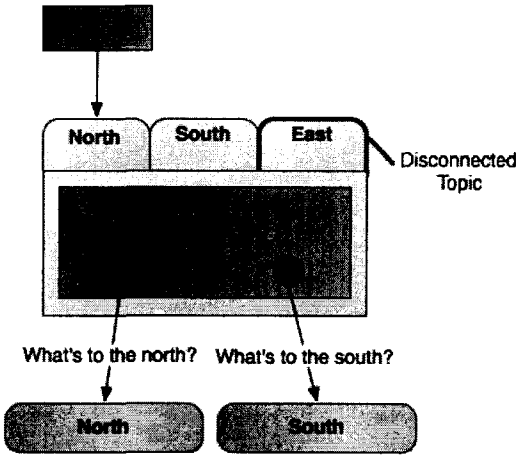


Figure 5.11: A disconnected topic in a topic group.

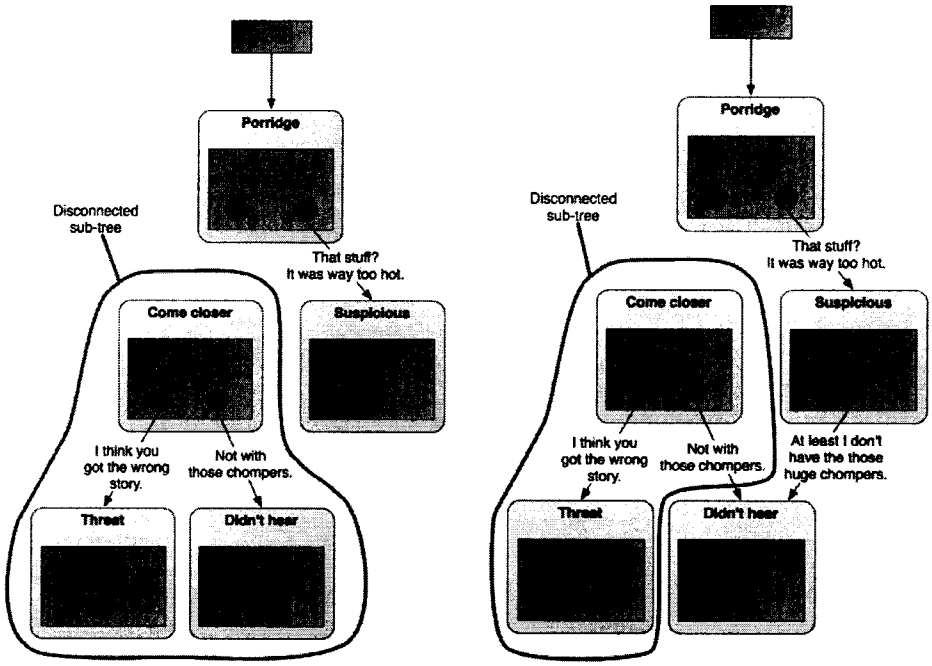
connected to the "Come closer" topic, they are still disconnected from the conversation. The "Come closer" topic is the root of a disconnected sub-tree and consequently all 3 topics are removed from the conversation. However, if the "Didn't hear" topic was also linked by the "Suspicious" topic as shown in Figure 5.12(b), then the "Didn't hear" topic is not disconnected and only the "Come closer" and "Threat" topics are removed.

The GUI can decide to delete disconnected sub-trees entirely, but it is advantageous to the author to store the removed sub-trees into a *disconnected bin*. The disconnected bin would treat removed sub-trees as individual dialogue pattern instances. Thus, the author can recover dialogue pattern instances from the disconnected bin by adding them back into the conversation using the *Add Dialogue Pattern* operation. After instantiation, the pattern instance is removed from the disconnected bin.

5.8 Decision Patterns

The author can instantiate a decision pattern using the **Add Decision Pattern** operation. The author must first select a precursor that will directly link to the new decision pattern. The GUI is responsible for providing a mechanism to select a specific decision pattern, similar to the encounter pattern picker window described in Chapter 2. If the outcome already links to an existing target, then the the link is redirected to the new decision pattern. The first outcome of the new decision pattern then links to the existing target. For example, in Figure 5.13 the author instantiates an *Ability decision* between the "Go away!" and "Sick" topics.

The author can also remove a decision pattern using the **Remove Decision Pattern** operation. Similar to removing a topic, any sub-trees linked by the removed decision that become disconnected are also removed.



(a) A disconnected sub-tree of 3 topics.

(b) A disconnected sub-tree of 2 topics. The "Didn't hear" topic remains connected.

Figure 5.12: Sub-trees disconnected from a conversation.

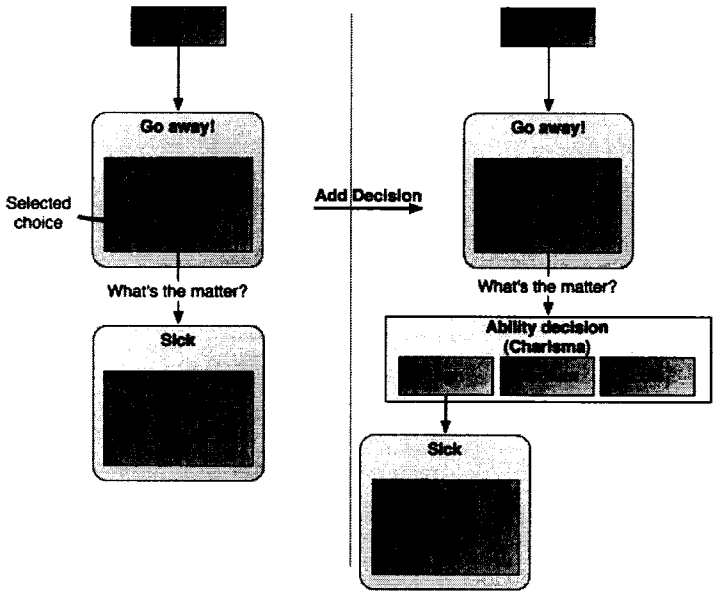


Figure 5.13: Adding an *Ability* decision pattern in a conversation.

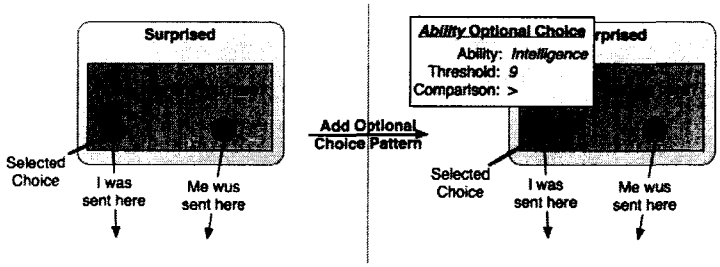


Figure 5.14: Adding an *Ability* optional choice pattern to a choice.

5.9 Optional Choice Patterns

The author can instantiate an optional choice pattern using the **Add Optional Choice Pattern** operation. The author must first select a choice. For example, in Figure 5.14 the author selects the "I was sent here" choice and uses the **Add Optional Choice Pattern** operation to instantiate a *Ability* optional choice pattern. The author then sets the pattern instance's options. The author can also attach several optional choice pattern instances to a single choice. For example, a second *Ability* optional choice pattern can be instantiated on the "I was sent here" choice. Now the conditions for both *Ability* optional choice pattern instances must be true in order for the choice to be available. To remove an optional choice pattern instance, the author can use the **Remove Optional Choice Pattern**. This operation does not remove the choice itself. An optional choice pattern instance can be attached to both choice singletons and choice groups. If the pattern instance is attached to a choice group, then the pattern condition applies to all PC remarks in the group.

5.10 Summary

This chapter described the operations used to construct conversations using dialogue patterns. The operations for topics, topic groups, exchanges, choices, linking, and dialogue patterns were described in order. Then, the semantics for removing a sub-tree from the conversation were described. Finally, the operations for instantiating decision and optional choice patterns were described.

Chapter 6

Evaluation - A Case Study

This chapter presents a case study to compare the conversation model supported by the Aurora conversation editor and the conversation model based on dialogue patterns. This case study uses four conversations in Chapter 1 of the Neverwinter Nights official campaign. The conversations vary in size and complexity, and are a good representative sample of all the conversations in Chapter 1 of the official campaign.

This chapter begins by defining five metrics that can be used to evaluate the complexity of conversations. The chapter then presents the four conversations that will be evaluated using the metrics. The chapter ends by applying the five metrics to the four conversations as built using the Aurora conversation model and the dialogue pattern model. The values of these complexity metrics are compared and these comparisons show that the dialogue pattern model has lower (better) values.

6.1 Complexity Metrics

There are two approaches to evaluate the effectiveness of the dialogue pattern model. In the first approach a tool is built that implements dialogue patterns. Then, a user study is conducted to measure the relative effectiveness of the Aurora conversation editor and the dialogue pattern tool. A group of game authors use both the Aurora conversation editor and the dialogue pattern tool. The authors then evaluate each tool based on a variety of criteria. Finally, comparisons are made between evaluations. If the dialogue pattern tool is evaluated to be better than the Aurora conversation editor, then a conclusion can be drawn that the dialogue pattern model is a better model than the model used by the Aurora conversation editor. The advantage this approach is that real-world experience is used to measure the effectiveness of the model. Ultimately, the goal of creating a new model is to provide a better mechanism for authors to create and edit conversations, and this is reflected in the data that would be produced from such a user study.

However, a tool that implements the dialogue pattern model is not available yet. It is not possible to compare both models using a user study, and therefore it is difficult to take into account flaws that are not exposed by users evaluating the program. Instead, the second approach is used. Rather than

comparing two specific tools that use the Aurora conversation model (Aurora conversation editor) and dialogue patterns (no tool yet), the models can be evaluated directly by applying complexity metrics to several conversations. In this dissertation, the term *complexity*[1] is used to describe the complicatedness of various aspects of a conversation model. The model with the lower overall complexity metric scores is considered less complex, and therefore is easier to use. The second approach also has a major advantage – the models are compared directly. It is possible that a poorly designed and implemented tool could disguise a very good model. The results from a user study ultimately reflect the quality of both the model and the tool implementing the model. A case study that measures the models directly is the best way to evaluate only the models themselves.

This section presents five metrics to measure conversation complexity: *component* complexity, *structural* complexity, *remark* complexity, *indirection* complexity, and *operational* complexity. The metrics are expressed formally as formulas and are denoted with a subscript $_A$ for the Aurora conversation model, and a subscript $_D$ for the dialogue patterns. A running example based on the "beggar1" conversation of chapter 1 of the official campaign is used to help illustrate how metrics are computed. This conversation has been modified slightly (by adding a single NPC remark) to better illustrate all the metrics.

6.1.1 Component Complexity

Component complexity measures the number of visible components when a conversation is fully expanded. In the case of the Aurora conversation model, this includes all NPC nodes, PC nodes, and link nodes. In the case of dialogue patterns, this includes all topics, choices, inner and tail exchanges, decision patterns, decision pattern outcomes, secondary links, and end dialogue targets.

For the Aurora conversation model, component complexity is defined in Figure 6.1. Component complexity is the number of all conversation nodes in the conversation. For example, there are 12 NPC nodes, 12 PC nodes, and 10 link nodes in the "beggar1" Aurora conversation in Figure 6.3. Therefore $component_A(beggar1) = 12 + 12 + 10 = 34$. The author views 34 total components when the conversation is fully expanded.

$$NPCNodes \stackrel{def}{=} \{NPC\ nodes\ in\ conversation\}$$
$$PCNodes \stackrel{def}{=} \{PC\ nodes\ in\ conversation\}$$
$$LNodes \stackrel{def}{=} \{link\ nodes\ in\ conversation\}$$
$$component_A(conversation) \stackrel{def}{=} |NPCNodes| + |PCNodes| + |LNodes|$$

Figure 6.1: Component complexity formula for Aurora conversations.

For dialogue patterns, component complexity is defined in Figure 6.2. In this case, component complexity is the sum of all topics, exchanges (inner and tail), choices, decision patterns (including degenerate), outcomes in decision patterns, secondary links, and end dialogue targets.

[1]The term *complexity* is not used here to refer to the space or time complexity of an algorithm, but rather how complicated an aspect of the model is.

All of these components, including inner exchanges, are visible when the conversation is fully expanded. For example, there are 9 topics, 10 exchanges, 9 choices, 4 decision patterns, 10 decision pattern outcomes, 3 secondary links, and 2 end dialogue targets in the "beggar1" dialogue pattern conversation in Figure 6.4. The 9 topics include the 3 topics in the topic group. Therefore $component_D(beggar1) = 9 + 10 + 9 + 4 + 10 + 3 + 2 = 47$. Comparing the two results shows that $component_D(beggar1)$ is 12 points higher than $component_A(beggar1)$.

This score is higher since topics, decision patterns, outcomes, and end dialogue targets are included in $component_D(beggar1)$ score and these components do not exist in the Aurora conversation model. Note that these components incorporate scripting and other information that is not represented in the Aurora conversation model. For example, the quest point decision in Figure 6.4 explicitly divides the conversation tree based on whether a quest has been completed. In the Aurora conversation model (in Figure 6.3) the author has no way of knowing that the remark on line 3 ("It's over?...") will only be displayed if the quest has been completed, without looking at the scripts. These extra visual components provide good added value. Each topic includes text that summarizes the intent of the topic. This information does not appear in the Aurora conversation model. The end dialogue markers could be excluded in favour of choices with no links to targets. However, it is easier for the author to spot these important locations in the conversation tree if explicit markers are used. If these value-added components are removed from the conversation, the ***reduced component complexity*** of the "beggar1" conversation would only be $10 + 9 + 3 = 22$.

$$
\begin{aligned}
&Topics \stackrel{def}{=} \{topics\ in\ conversation\}\\
&TExchanges \stackrel{def}{=} \{tail\ exchanges\ in\ conversation\}\\
&IExchanges \stackrel{def}{=} \{inner\ exchanges\ in\ conversation\}\\
&Exchanges \stackrel{def}{=} TExchanges \cup IExchanges\\
&TChoices \stackrel{def}{=} \{choices\ in\ tail\ exchanges\ in\ conversation\}\\
&IChoices \stackrel{def}{=} \{choices\ in\ inner\ exchanges\ in\ conversation\}\\
&Choices \stackrel{def}{=} TChoices \cup IChoices\\
&Decisions \stackrel{def}{=} \{decision\ pattern\ instances\ in\ conversation\}\\
&Outcomes_n \stackrel{def}{=} \{outcomes\ in\ decision\ pattern\ \mathbf{n}\}\\
&LSec \stackrel{def}{=} \{secondary\ links\ in\ conversation\}\\
&EDTargets \stackrel{def}{=} \{end\ dialogue\ targets\ in\ conversation\}\\
&component_D(conversation) \stackrel{def}{=} |Topics| + |Exchanges| + |Choices| + |Decisions| \\
&\qquad\qquad + \sum_{n \in Decisions} |Outcomes_n| + |LSec| + |EDTargets|\\
&reduced\_component_D(conversation) \stackrel{def}{=} |Exchanges| + |Choices| + |LSec|
\end{aligned}
$$

Figure 6.2: Component complexity formulas for dialogue patterns.

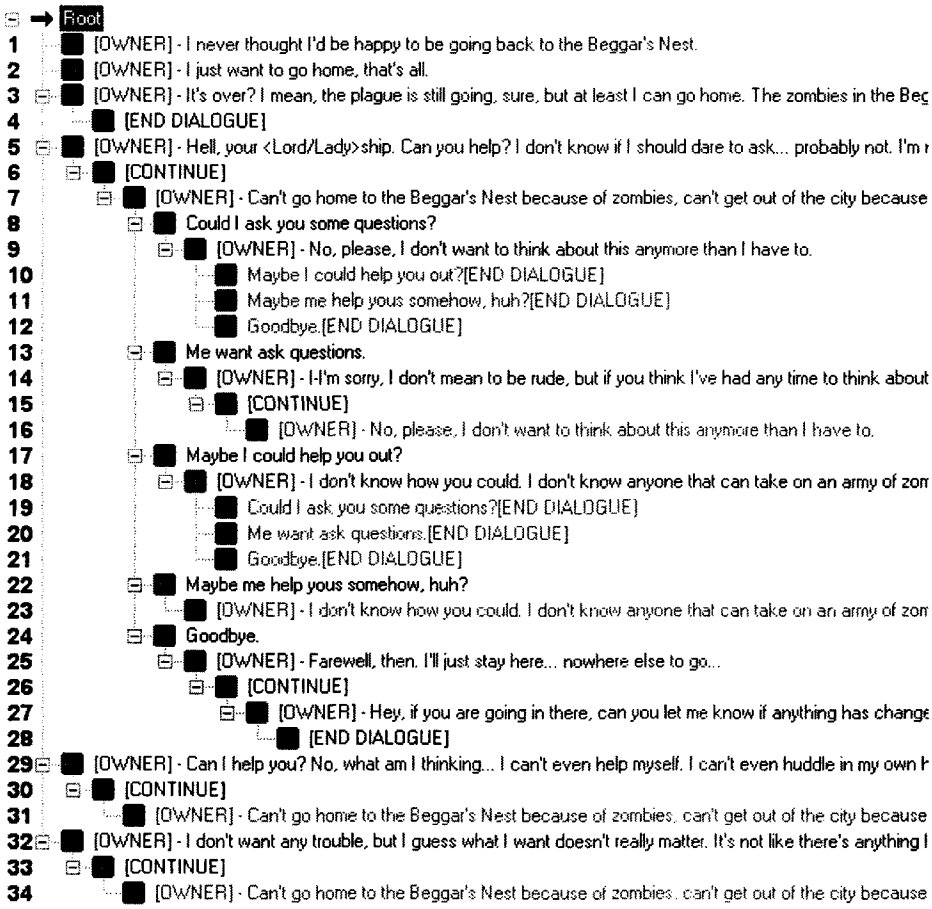


Figure 6.3: The "beggar1" conversation in the Aurora conversation editor.

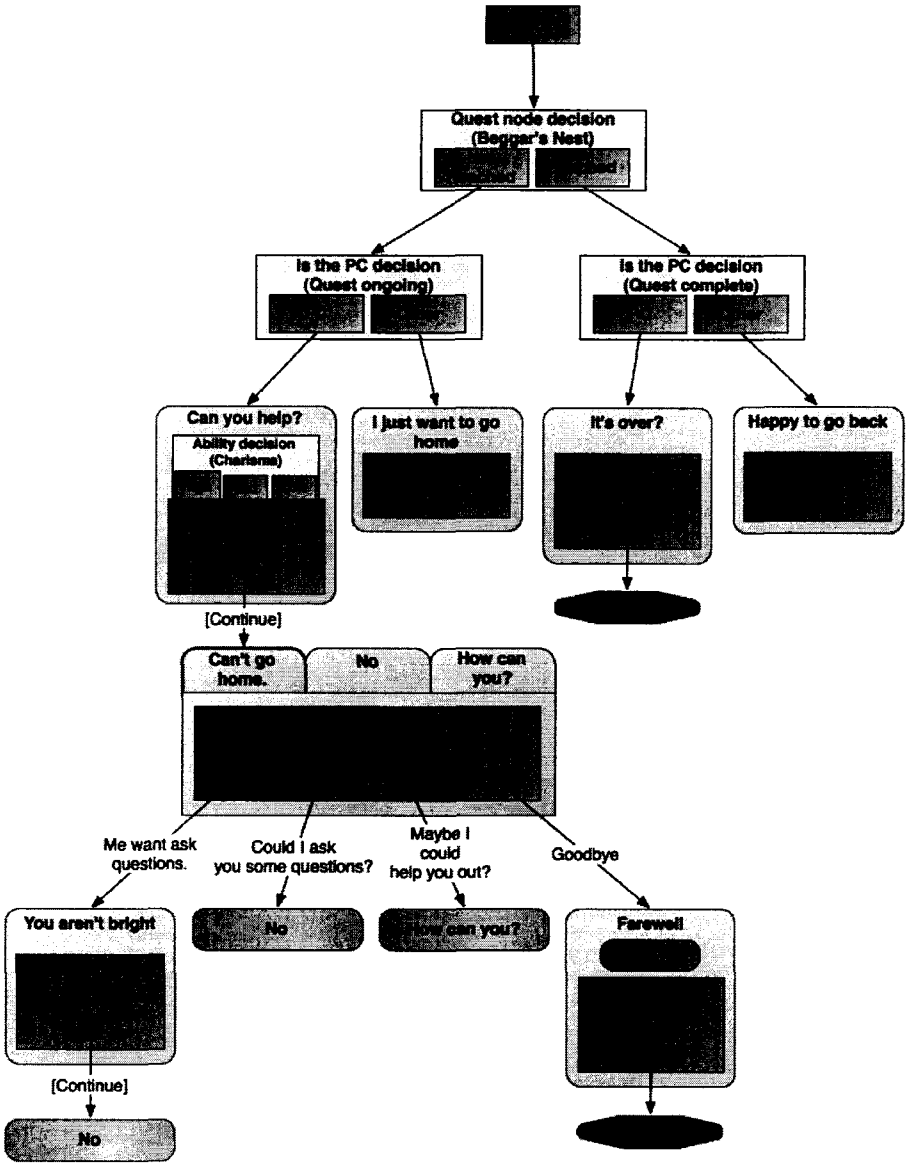


Figure 6.4: The "beggar1" conversation in the dialogue pattern model.

6.1.2 Structural Complexity

Structural complexity measures the number of visible components when a conversation has hidden all components that are unnecessary for the author to understand the intent of the conversation. In the case of the Aurora conversation model, this metric is the same as component complexity since it is not possible to collapse any sub-tree without losing important information such as branching factor or the target of link nodes. The Aurora conversation editor could be modified so that any linear chains of conversation nodes could be collapsed to show only the first and last node in the chain. For example, in Figure 6.3 lines 5, 6, 7, 8, and 9 could be collapsed to show only lines 5 and 9. In the case of the dialogue pattern model, this metric assumes that all topics with multiple exchanges are collapsed to show only the tail exchange. Therefore inner exchanges and the choices inside these exchanges are not counted.

For the Aurora conversation model, structural complexity is defined in Figure 6.5. Similar to component complexity, structural complexity is the sum of all conversation nodes in the conversation. For example, there are 12 NPC nodes, 12 PC nodes, and 10 link nodes in the "beggar1" Aurora conversation in Figure 6.3. Therefore $structural_A(beggar1) = 12 + 12 + 10 = 34$. If the Aurora conversation were modified to collapse linear chains then 10 nodes could be hidden and therefore a modified structural complexity metric called *collapsed chain structural complexity* would be 24. However, since the Aurora conversation editor cannot collapse chains of nodes, the *collapsed chain structural complexity* is not included in the results in Section 6.3.

$$structural_A(conversation) \stackrel{def}{=} |NPCNodes| + |PCNodes| + |LNodes|$$

Figure 6.5: Structural complexity formula for Aurora conversations.

For dialogue patterns, structural complexity is defined in Figure 6.6. In this case, structural complexity is the total number of components, as defined for component complexity, minus the inner exchanges and choices inside inner exchanges. For example, there are 9 topics, 9 exchanges, 8 choices, 4 decision patterns, 10 decision pattern outcomes, 3 secondary links, and 2 end dialogue targets in the "beggar1" dialogue pattern conversation in Figure 6.4. The 9 topics include the 3 topics in the topic group. Therefore $structural_D(beggar1) = 9+9+8+4+10+3+2 = 45$. Comparing the two results shows that $structural_D(beggar1)$ is 11 points higher than $structural_A(beggar1)$. Again this score is higher since since value-added components such as topics, decision patterns, outcomes, and end dialogue targets are included in $structural_D(beggar1)$ score. Without these value-added components, the *reduced structural complexity* score is $9 + 8 + 3 = 20$.

6.1.3 Remark Complexity

Remark complexity measures the number of remarks for which the author enters text when building the conversation. This metric is useful since it directly measures the number of lines of conversation the author must write. This metric can be reduced by reusing the same lines of text in several

$$structural_D(conversation) \stackrel{def}{=} |Topics| + |TExchanges| + |Choices| + |Decisions| + \sum_{n \in Decisions} |Outcomes_n| + |LSec| + |EDTargets|$$

$$reduced\_structural_D(conversation) \stackrel{def}{=} |TExchanges| + |TChoices| + |LSec|$$

Figure 6.6: Structural complexity formulas for dialogue patterns.

contexts of the conversation. In the Aurora conversation model, this is done by creating a link node to an existing NPC or PC conversation node. In dialogue patterns, text for PC remarks is reused by sharing choices between topics in a topic group.

For the Aurora conversation model, remark complexity is defined in Figure 6.7. The difference between structural and remark complexity is that link nodes are not counted since the author does not need to write text for the link nodes. A link node automatically uses the text of its target. The remark complexity can be reduced by replacing duplicated NPC or PC nodes with link nodes. In the "beggar1" Aurora conversation (Figure 6.3) there are 12 NPC nodes and 12 PC nodes. Consequently $remark_A(beggar1) = 12 + 12 = 24$. This means the author has to set the text for 24 remarks.

$$remark_A(conversation) \stackrel{def}{=} |NPCNodes| + |PCNodes|$$

Figure 6.7: Remark complexity formula for Aurora conversations.

For dialogue patterns, remark complexity is defined in Figure 6.8. Recall that each exchange has one NPC remark, each choice singleton has one PC remark, and each choice group has a number of PC remarks equal to the number of choices in the group. Therefore, remark complexity is the sum of all exchanges, all choice singletons, and all choices in choice groups in the conversation. The number of PC remarks in a choice is represented by PCr_n. If n is a choice singleton, PCr_n is 1. For example, the "beggar1" conversation in Figure 6.4 has 11 tail exchanges, 1 inner exchange that includes 1 hidden choice singleton, 7 choice singletons, and 1 choice group that contains 2 PC remarks. The 3 tail exchanges in the "Can you help?" are included in the 11 tail exchanges. Therefore $remark_D(beggar1) = 11 + 1 + 1 + 7 + 1 \times 2 = 22$. Therefore, the author has to set the text for 22 remarks.

This result differs from the remark complexity score of 24 for the Aurora version of the conversation. How can the two conversations be equivalent yet have a difference of two remarks? The number of remarks the author needs to set depends on the number of duplicated nodes. For example, in the Aurora "beggar1" conversation in Figure 6.3 the PC nodes on lines 6, 30, and 33 are duplicates since they link to the same NPC node and have the same remark text. The author could have replaced the PC nodes on lines 30 and 33 with link nodes pointing to the PC node on line 6. In contrast, the "beggar1" dialogue pattern has no duplication in this case since it shares only one choice for all three exchanges in the "Can you help?" topic.

Note there is no *reduced remark complexity* for dialogue patterns since none of the value-added

components contain any remarks. Similarly, if the Aurora conversation editor had the capability to collapse linear chains, the collapse would not reduce the number of remarks and therefore there is no *collapsed chain remark complexity* metric.

$$PCr_n \stackrel{def}{=} \{PC\ remarks\ in\ choice\ \mathbf{n}\}$$
$$remark_D(conversation) \stackrel{def}{=} |Exchanges| + \sum_{n \in Choices} |PCr_n|$$

Figure 6.8: Remark complexity formulas for dialogue patterns.

6.1.4 Indirection Complexity

Indirection complexity measures the number of points in the conversation where the conversation tree is terminated with a leaf node that acts as a placeholder for another component. In the Aurora conversation model this is a link node that points to an NPC or PC node. In the dialogue pattern it is a secondary link which is indicated by a link target. This metric measures the disjointedness of the tree where the author must follow the link by changing the view to the link target. It is easy to keep this metric low by replacing existing links with duplicated sub-trees. However, this duplication forces the author to enter more text for remarks, and therefore causes the remark complexity to increase. The goal of the dialogue pattern model is to minimize indirection complexity without greatly increasing remark complexity.

For the Aurora conversation model, indirection complexity is defined in Figure 6.9. In this case, indirection complexity is the number of link nodes in the conversation. Each link node is a terminal node and forces the author to change the viewport to find the link's target node. For example, the "beggar1" conversation in Figure 6.3 has 10 link nodes and therefore $indirection_A(beggar1) = 10$. There are 10 points in the conversation where the author's view must be redirected.

$$indirection_A(conversation) \stackrel{def}{=} |LNodes|$$

Figure 6.9: Indirection complexity formula for Aurora conversation model.

For dialogue patterns, indirection complexity is defined in Figure 6.10. In this case, indirection complexity is the number of secondary links in the conversation. Each secondary link has a link target redirecting the author to another part of the conversation. For example, the "beggar1" dialogue pattern conversation in Figure 6.4 has 3 secondary links and therefore $indirection_D(beggar1) = 3$. This conversation has a lower disjointedness than the Aurora counterpart without increasing the remark complexity.

$$indirection_D(conversation) \stackrel{def}{=} |LSec|$$

Figure 6.10: Redirection complexity formula for dialogue patterns.

6.1.5 Operational Complexity

Operational complexity measures how many operations the author must perform to construct the conversation. For simplicity, all operations have a cost of one unit. This metric only counts the *minimum* number of operations needed to construct the conversation. For example, if a conversation in the dialogue pattern model has five topic groups with three topics per group, the conversation is built with 15 *Add Topic* operations and five *Merge Topic* operations for a total of 20 operations. For each topic group, the first two topics are created separately. They are then merged with the *Merge Topic* operation. The third topic is created directly in the topic group with the *Add Topic* operation. It is possible to create the same five topic groups with more operations by creating all 15 topics separately with 15 *Add Topic* operations. Then each of the topic groups could be created by using two *Merge Topic* operations to merge three topics together. This method would result in a total of 25 operations. Therefore it is more efficient to add extra topics directly in the topic group.

For the Aurora conversation model, operational complexity is defined in Figure 6.11. In this case, operational complexity includes one **Add Node** operation for each NPC and PC node, one **Edit Text** operation for each NPC and PC node, and one **Add Link** operation for each link node. Additionally, the metric includes one **Write Script** operation for each unique script (**When** or **What**) used by the conversation and one **Attach Script** operation for each script that is attached to each node. For example, if a node has both a **When** and a **What** script attached then two **Attach Script** operations are counted. In this case, the node is said to have two attach points, one for each script. The "beggar1" Aurora conversation in Figure 6.3 has 12 NPC nodes, 12 PC nodes, 10 link nodes, and uses 8 unique scripts on 10 attach points. Therefore $operation_A(beggar1) = 12 + 12 + 12 + 12 + 10 + 8 + 10 = 76$. The author needs to perform 76 operations to construct this conversation.

For simplicity, writing a script is counted as one operation, even though writing a script is much more complex than writing text for a single node. Similarly, adding an adapted decision or optional choice pattern in a dialogue pattern conversation is counted as one operation, regardless of the number of adaptations. This simplifies the metric without having to introduce weights for each operation. More complex metrics could be constructed by count lines of code in a script and adaptation to a pattern instance. However, in this case it is not clear how to weight lines of code versus adaptations.

$$NAdd \stackrel{def}{=} NPCNodes \cup PCNodes$$
$$TEdit \stackrel{def}{=} NPCNodes \cup PCNodes$$
$$LAdd \stackrel{def}{=} \{link\ nodes\ in\ conversation\}$$
$$SWrite \stackrel{def}{=} \{unique\ scripts\ in\ conversation\}$$
$$SAttach \stackrel{def}{=} \{attached\ scripts\ in\ conversation\}$$
$$redirection_A(conversation) \stackrel{def}{=} |NAdd| + |TEdit| + |LAdd| + |SWrite| + |SAttach|$$

Figure 6.11: Operation complexity formulas for Aurora conversations.

For dialogue patterns, operational complexity is defined in Figure 6.12. In this case, operational

complexity includes the minimum number of operations described in Chapter 5 to construct the conversation. The **Add Topic** operation is counted for each topic in the conversation. One **Merge Topic** operator is counted for each topic group.

Operational complexity also counts one **Insert Exchange** operation for each inner exchange in the conversation. The **Add Decision Pattern** and **Add Decision Pattern as Degenerate** operations are counted for each decision pattern and degenerate decision pattern, respectively. The **Remove Choice** is counted for each utterance topic, since an utterance is created by removing the last choice in a topic. The **Add Choice Singleton** is counted for each choice singleton after the first choice in each topic. The first choice in a topic is created along with the topic, and therefore is not counted. Similarly, the **Add Choice Group** operation is counted for each choice group added after the first choice in each topic. Finally, the **Convert Singleton to Group** is counted for each topic that has a choice group as the first choice.

The **Link** operator is counted once for each secondary link, and each direct link that was not created implicitly with a **Add Topic** or **Add Decision Pattern** operation. Additionally, the **Set Remark Text** operation is counted once for each NPC and PC remark in the conversation. This is simply the value of the conversation's remark complexity. Finally, the **Add Optional Choice Pattern** and **Add *What* Encounter Pattern** operations are counted once for each optional choice pattern and external **What** encounter pattern, respectively. Although **What** encounter patterns are not part of dialogue patterns, they are included for fairness since the operational complexity for the Aurora conversation model counts attached **What** scripts. If a decision pattern includes **What** actions on a remark, those **What** actions are not counted as a **What** encounter pattern since they are generated by the decision pattern.

For the "beggar1" dialogue pattern conversation in Figure 6.4 the operations are shown in Table 6.1. The 9 **Add Topic** operations add the four topics on row three in Figure 6.4, the three topics in the topic group on row four, and the two topics on row five. The one **Merge Topic** operation creates the topic group on row four that contains the "Can't go home.", "No", and "How can you?" topics. The one **Insert Exchange** operation creates the inner exchange in the "Farewell" topic on row five. The three **Add Decision Pattern** operations add the three decision patterns found on rows one and two. The **Add Decision Pattern as Degenerate** operation adds the degenerate decision found in the "Can you help?" topic on row three. The two **Add Choice Singleton** operations add two choice singletons to the topic group on row four: the "Could I ask you some questions?" choice singleton (position 2) and the "Goodbye" choice singleton (position 4). The two **Remove Choice Singleton** operations remove choices from the "I just want to go home" and "Happy to go back" utterance topics on row three. The one **Add Choice Group - *Intelligence*** operation adds the "Maybe I could help you out?" choice group (position 3) in the topic group on row four. The three **Link** operations create the three secondary links found on rows five and six. The 22 **Set Remark Text** operations set all the remark text found in the conversation: 12 exchanges (each with an NPC remark), 8 choice singletons (each

with one PC remark), and one choice group (two PC remarks). The two **Add Optional Choice Pattern** operations add one optional choice pattern to both the "Me want ask questions" (position 1) and "Could I ask you some questions?" (position 2) choice singletons in the topic group on row four. Finally, the one **Add What Encounter Pattern** operation adds an external encounter pattern that performs some actions at a point in the conversation.

Summing up the rows results in a operational complexity of $operational_D(beggar1) = 49$. Assuming that writing a script and adapting a pattern to be equal in cost, 28 fewer operations are used to build the conversation with the dialogue pattern model compared to the Aurora conversation model. In some sense, adapting a pattern is like writing a script in a higher level language.

Operation	Count
Add Topic	9
Merge Topic	1
Insert Exchange	1
Add Decision Pattern	3
Add Decision Pattern as Degenerate	1
Add Choice Singleton	2
Remove Choice Singleton	2
Add Choice Group - *Intelligence*	1
Convert Choice to Group - *Intelligence*	0
Link	3
Set Remark Text	22
Add Optional Choice Pattern	3
Add **What** Encounter Pattern	1

Table 6.1: Number of operations used to build the "beggar1" conversation using the dialogue pattern model.

6.2 The Conversations

This sections describes the four conversations selected for the case study. The *Emernick*, *Nurse*, *Helmite*, and *Bertrand* conversations are from Chapter 1 of the Neverwinter Nights official campaign. For the Aurora version of each conversation, the **When** and **What** scripts are briefly summarized. Then the dialogue pattern version of the conversation is described and the decision and optional choice pattern instances are identified. This section presents figures for each conversation under the dialogue pattern model. Unfortunately conversations under the Aurora conversation model are too large to display in this dissertation and therefore are not included. However, the complexity metrics for each Aurora conversation are given in Section 6.3.

The majority of optional choice pattern instances are either a *Normal intelligence* or *Low intelligence* optional choice pattern. These two patterns are specializations of the *Ability* optional choice pattern and are described in Appendix A. All *Normal intelligence* optional choices are available when the PC has intelligence greater than 9. All *Low intelligence* optional choices are available when the PC has intelligence of 9 or lower. These patter instances are automatically attached to

$$
\begin{aligned}
&TopAdd \stackrel{def}{=} \{topics\ in\ conversation\}\\
&TopMerge \stackrel{def}{=} \{topic\ groups\ in\ conversation\}\\
&EInsert \stackrel{def}{=} \{inner\ exchanges\ in\ conversation\}\\
&DPAdd \stackrel{def}{=} \{normal\ decision\ patterns\ in\ conversation\}\\
&DDPAdd \stackrel{def}{=} \{degenerate\ decision\ patterns\ in\ conversation\}\\
&CAdd \stackrel{def}{=} \{choices\ in\ conversation\}\\
&CRem \stackrel{def}{=} \{utterances\ in\ conversation\}\\
&GAdd \stackrel{def}{=} \{choice\ groups\ in\ conversation\}\\
&GConvert \stackrel{def}{=} \{converted\ choices\ in\ conversation\}\\
&Link \stackrel{def}{=} \{secondary\ links\ in\ conversation\}\\
&TSet \stackrel{def}{=} remark_D(conversation)\\
&OCPAdd \stackrel{def}{=} \{optional\ choice\ patterns\ in\ conversation\}\\
&WAdd \stackrel{def}{=} \{external\ what\ patterns\ associated\ with\ conversation\}\\
&operational_D(conversation) \stackrel{def}{=} |TopAdd| + |TopMerge| + |EInsert| +\\
&\qquad |DPAdd| + |DDPAdd| + |CAdd| +\\
&\qquad |CRem| + |GAdd| + |GConvert| +\\
&\qquad |Link| + |TSet| + |OCPAdd| + |WAdd|
\end{aligned}
$$

Figure 6.12: Operation complexity formulas for dialogue patterns.

choices in a *Intelligence* choice group.

6.2.1 Emernick

Emernick is an NPC that is trapped in the Neverwinter prison. He speaks a few one-liners to coax the PC into a saferoom. External scripts trigger these one-liners and require no intervention from the PC. Once safely inside, he is willing to converse with the PC and answer a few questions. At the end of the conversation, the PC can order Emernick to follow closely or stay in the saferoom.

Under the Aurora conversation model, the Emernick conversation has 8 unique **When** scripts attached to 43 nodes. The conversation has 4 of these scripts attached to 4 NPC nodes and are used to check if Emernick is safely inside the saferoom. The other 4 **When** scripts are attached to 38 PC nodes. Of these 38 PC nodes, 18 have the "nw_d2_intl" script attached to check if the PC has a low intelligence score. Another 18 PC nodes have the "nw_d2_intn" script attached to check if the PC has a normal or higher intelligence score. The other 2 unique **When** scripts combine either the normal or low checks for intelligence with a condition checking whether the PC has a high wisdom ability score.

Additionally, the Emernick conversation also has 5 unique **What** scripts attached to 6 nodes. Of these 5 scripts, 3 scripts control Emernick's movement by either following the PC, staying still, or moving towards the saferoom. The other 2 scripts are used if the PC attacks Emernick. One script engages Emernick in combat with the PC and the other script shifts the PC's alignment towards evil.

The dialogue pattern version of the Emernick conversation is shown in 2 different parts in Figures

6.13 and 6.14. The red arrow symbol indicates where the parts join to form the entire conversation.

The Emernick conversation (Figures 6.13 and 6.14) has 4 decision pattern instances and 1 optional choice pattern instance. There is one instance for each of the *Quest point*, *Progress*, *Door locked*, and *Near by* decision patterns. The 1 optional choice pattern instance is an *Ability* optional choice pattern. These patterns are describe in Appendix A.

The *Peninsula quest closed* pattern instance (*Quest point* decision) is used to determine if Emernick will converse with the PC. If the Peninsula quest is unfinished, the PC can start a conversation with Emernick. If the quest is finished, Emernick will just speak a one-liner. For the "Closed" outcome, the **Quest Point** option is set to the closed state of the Peninsula quest and the **Point State** option is set to *Reached*.

The *Saferoom door* pattern instance (*Door locked* decision) determines whether Emernick is securely inside the saferoom. If the door is locked, it is assumed that both the PC and Emernick are inside. In this case, the PC can start a conversation with Emernick. The pattern instance's **Door** global option is set to the *Saferoom Door* door object.

If the saferoom door is unlocked, then the *Saferoom* pattern instance (*Near by* decision) determines whether Emernick is inside or outside the saferoom by comparing the distance between Emernick and a waypoint inside the saferoom against a distance threshold. If Emernick is within 3 meters of the waypoint, he is considered to be inside the room. The pattern instance's **First Object** global option is set to *Emernick*, the **Second Object** global option is set to *Saferoom Waypoint*, and the **Distance** option is set to 3 meters.

Once secured inside the saferoom, Emernick is willing to speak to the PC. The *First time* pattern instance (*Progress* decision) ensures that Emernick's question about the PC's loyalty is only asked once. The **Goal** option for the "First" outcome is set to the NPC remark in the "Not with prisoners?" topic.

The Emernick conversation only has one instance of the *Ability* optional choice pattern. This instance is attached to the "[Insight] How many levels?" choice in the topic group that contains the "Head Gaoler?" topic. This choice is only available if the PC has high wisdom. The pattern instance's **Ability** option is set to *wisdom*, the **Threshold** option is set to 14, and the **Comparison** option is set to ">" (i.e. greater than).

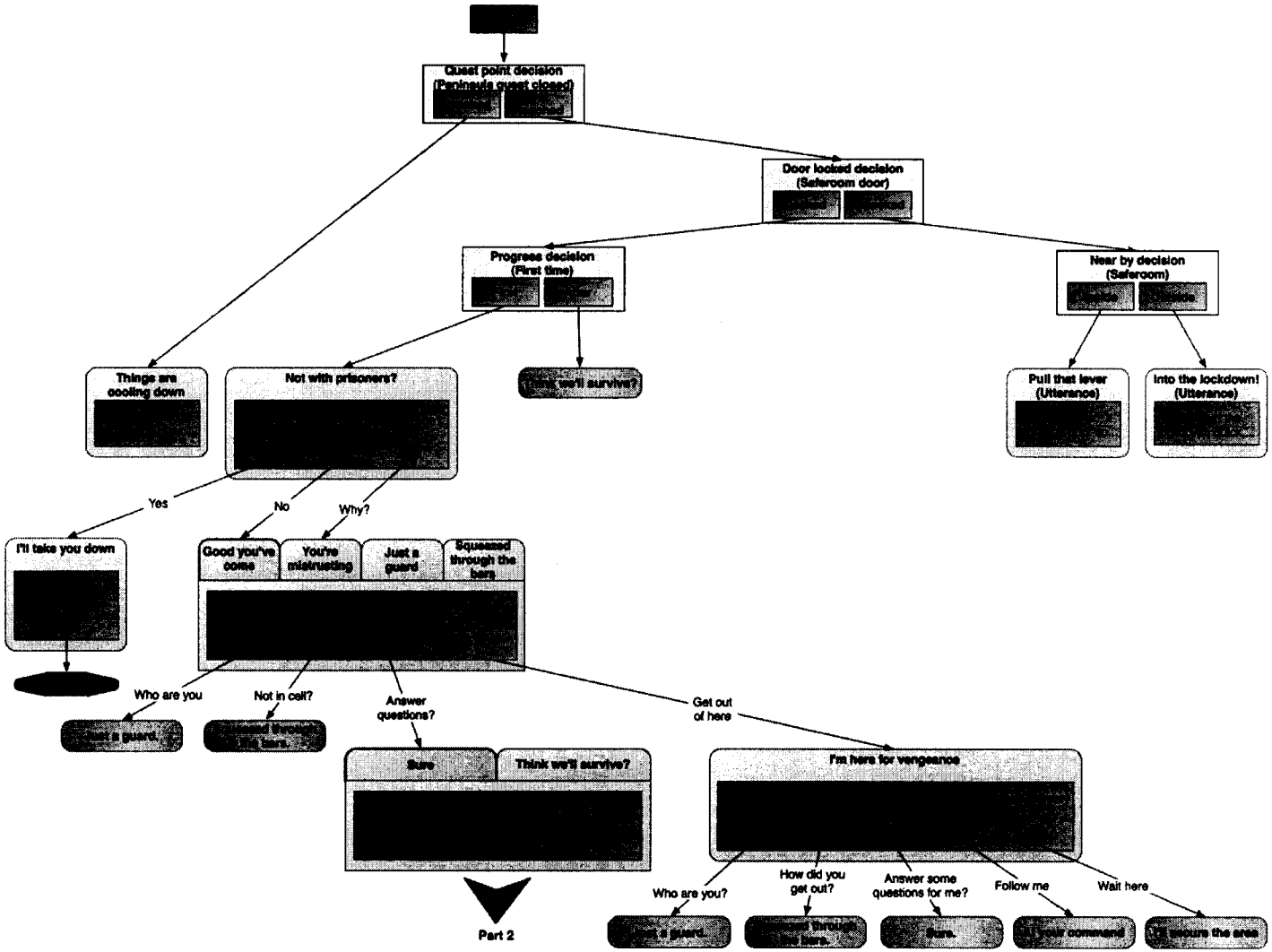


Figure 6.13: The Emernick conversation in the dialogue pattern model (Part 1).

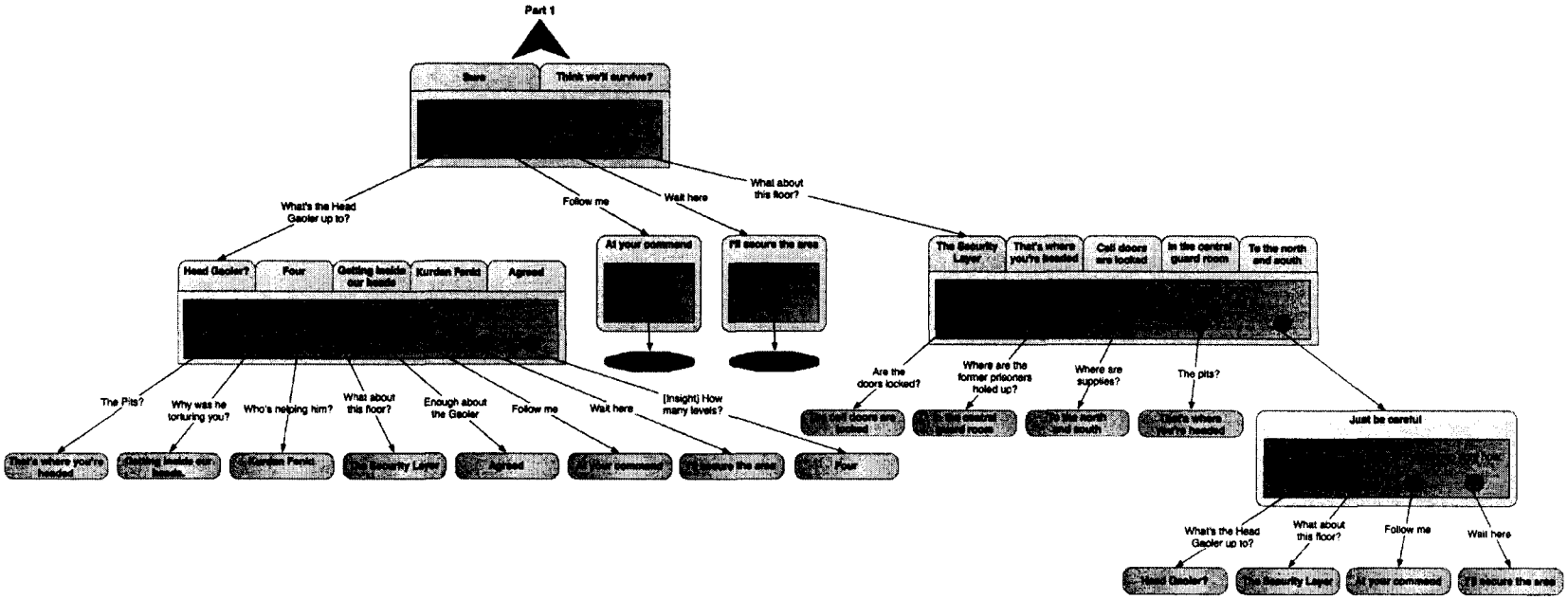


Figure 6.14: The Emernick conversation in the dialogue pattern model (Part 2).

6.2.2 Nurse

The Nurse conversation is used for several nurse NPCs that are tending wounded civilians when the PC first starts Chapter 1. These NPCs speak briefly with the PC and answer some basic questions. The conversation is identical for each nurse NPC.

Under the Aurora conversation model, the Nurse conversation has 8 unique **When** scripts attached to 38 nodes. Of these 8 scripts, 3 scripts are attached to 5 NPC nodes. The first script is used for a persuasion skill check to convince the NPC to tell the PC more gossip. The other 2 unique scripts are used on 4 NPC nodes and check if the PC has a high or normal charisma ability score. These scripts are used to change how the NPC reacts to the PC's appearance.

The other 5 **When** scripts are attached to 33 PC nodes. Of these 33 PC nodes, 15 have the "nw_d2_intl" script attached to check the PC for a low intelligence ability score. Another 15 PC nodes have the "nw_d2_intn" script attached to check the PC for a normal intelligence score. Another 2 unique **When** scripts combine either the normal or low conditions for intelligence with a condition checking whether the PC has at least a normal charisma ability score. The final 2 unique **When** script combine either the normal or low checks for intelligence with a condition checking whether the PC has a high wisdom ability score. The Nurse conversation does not have any **What** scripts.

The Nurse conversation built with the dialogue pattern model is shown in 2 different parts in Figures 6.15 and 6.16. The red arrow symbol indicates where the parts join to form the entire conversation. This conversation has 4 decision pattern instances and 10 optional choice pattern instances. The 4 decision pattern instances are instantiated from 2 decision patterns: *Ability* and *Skill* decision patterns. The 10 optional choice pattern instances are instantiated from 3 optional choice patterns: *Low Intelligence*, *High Intelligence*, and *Ability* optional choice patterns. These patterns are described in Appendix A.

There is one instance of the *Skill* decision pattern. The *Persuade - Gossip* instance makes an easy *persuasion* skill check to determine if the NPC is willing to tell the PC more gossip. The **Skill** global option is set to *persuasion* and the **Difficulty** global option is set to *Easy*.

There are 3 identical instances of the *Ability* decision pattern. The 3 *Charisma* instances are degenerate decisions inside the "Greetings", "Questions", and "Goodbye" topics. These decision instances decide on an NPC remark based on whether the PC has low, medium or high charisma. Regardless of the PC's charisma, the same set of choices are available to the PC. For each instance, the global **Ability** option is set to *charisma* and the **Threshold** option for the "High" outcome is set to 14. The **Threshold** option for the "Med" outcome is set to 9. The *Ability* decision instance inside the "Questions" is of particular interest since it is inside a topic group. The pattern instance is placed directly under the "Questions" tab to indicate it is not shared by the other topics in the group.

There are 5 instances of the *High Intelligence* optional choice pattern and 4 instances of the *Low Intelligence* optional choice pattern. These instances are attached to choice singletons. The choice singletons cannot be grouped into *Intelligence* choice groups since the NPC responds differently

depending on the PC's intelligence. Consequently a low intelligence choice singleton links to a different target than the corresponding high intelligence choice singleton. For example, the "Can I get directions?" and the "Me need directions" choices in the "Questions" topic are an intelligence pair. They are not a choice group since the "Can I get direction?" choices links to the "Not a tour guide" topic and the "Me need directions" choice links to the "Snipe comment" topic. The fifth *High Intelligence* pattern instance is attached to the "[Insight] You don't like Desther?" choice in the "Attack on Academy" topic. This choice does not have a corresponding low intelligence choice.

There is also one instance of the *Ability* optional choice pattern. This instance is also attached to the "[Insight] You don't like Desther?" choice in the "Attack on Academy" topic. The **Ability** option is set to *wisdom*, the **Threshold** option is set to *14*, and the **Comparison** option is set to $>$ (greater than). Therefore, this choice is only available if the PC has normal intelligence *AND* has high wisdom.

The *Nurse* conversation also has special notation in Part 1 (Figure 6.15). There is a topic group containing the "Plague", "Any more questions?", "Cure", and "I was a midwife" topics. Both the "Cure" and "I was a midwife" topics have their inner exchange indicator marked with a choice group symbol labeled "1". This indicates that an inner exchange has a choice group instead of a choice singleton. These choice groups are accounted in the *Nurse* metric calculations in Section 6.3.

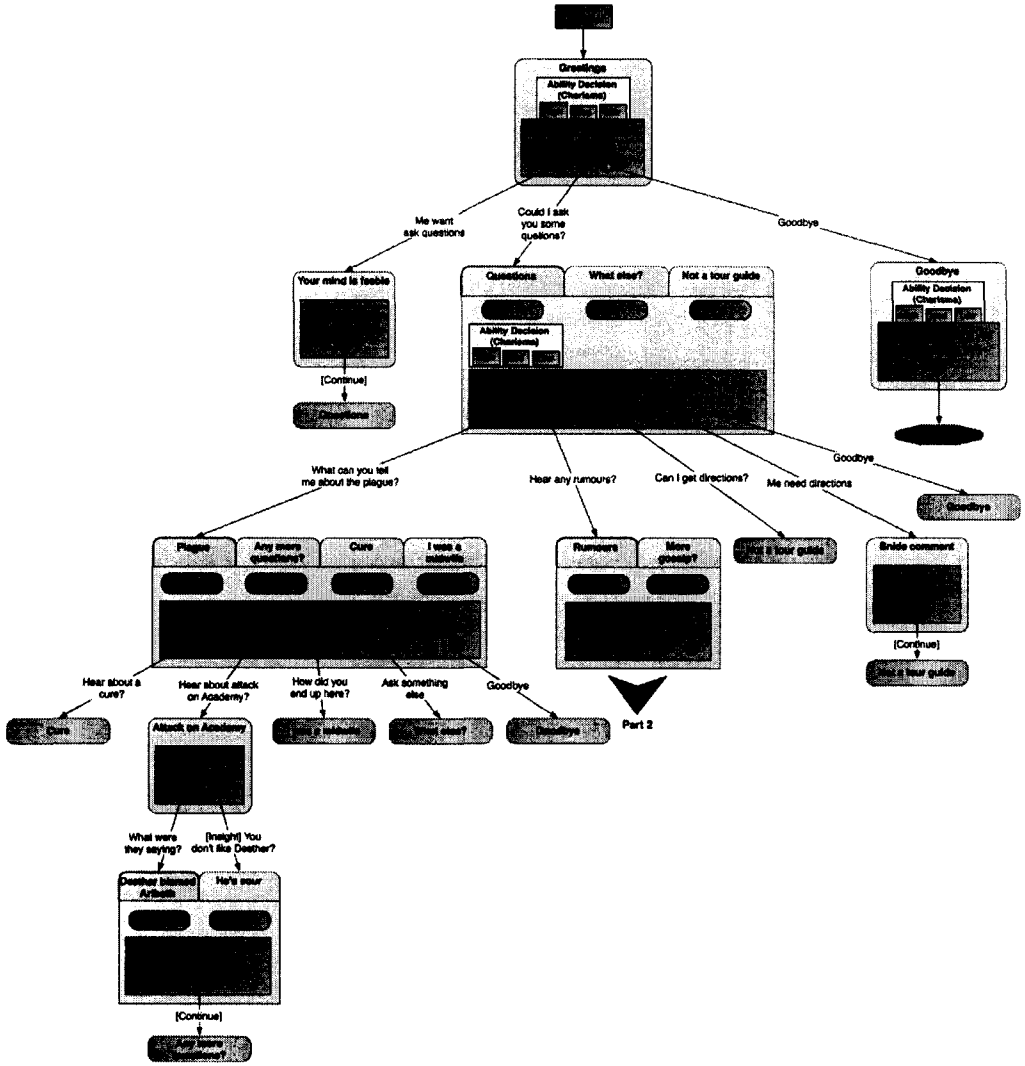


Figure 6.15: The Nurse conversation in the dialogue pattern model (Part 1).

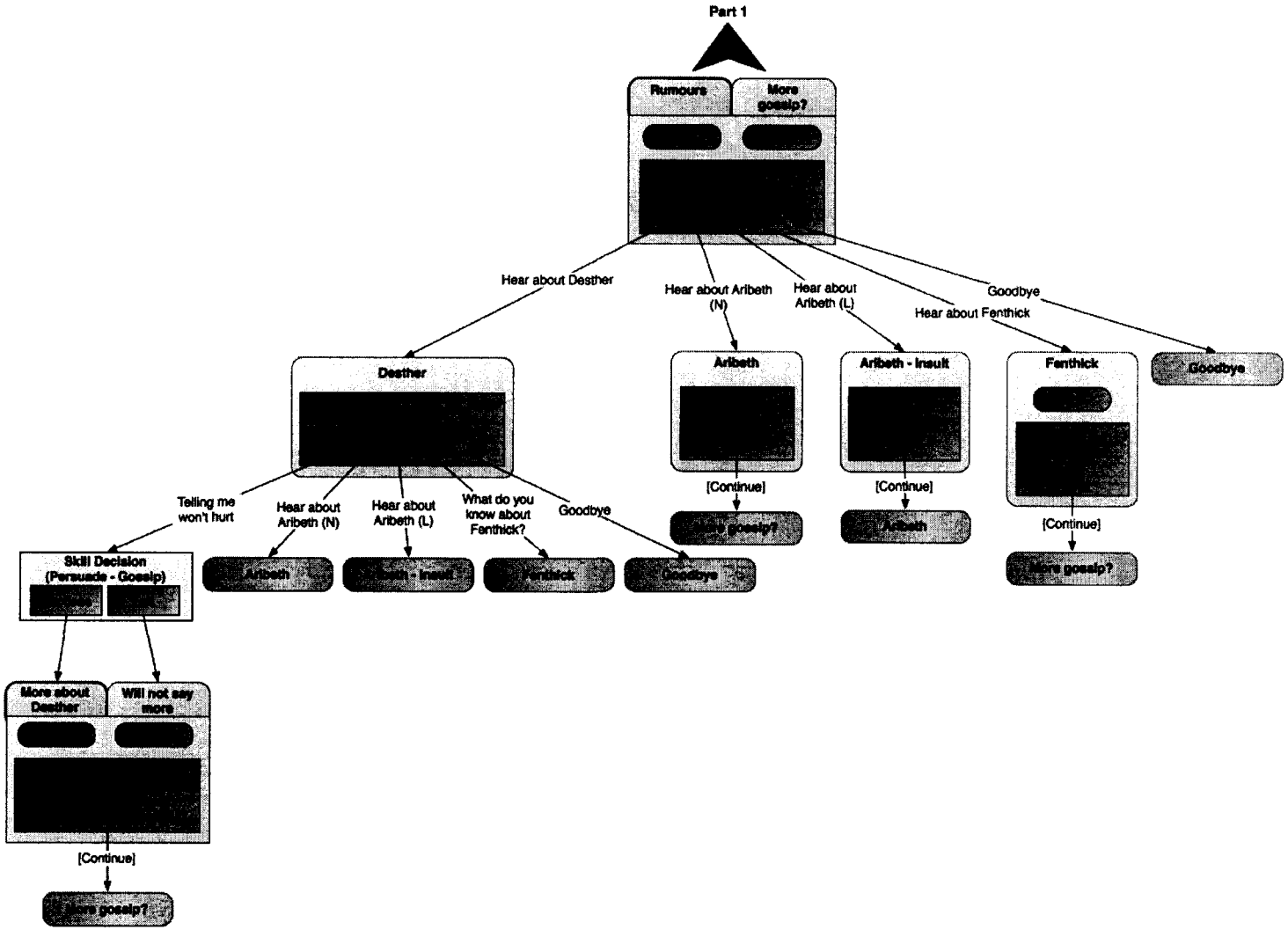


Figure 6.16: The Nurse conversation in the dialogue pattern model (Part 2).

6.2.3 Helmite

The Helmite conversation is used for a helmite[2] cleric NPC. The PC can ask the cleric questions, or receive a blessing.

Under the Aurora conversation model, the Emernick conversation has 7 unique **When** scripts attached to 25 nodes. Of these 7 scripts, 2 scripts are attached to 4 NPC nodes. The first script is used for the NPC's greeting and checks whether the PC has already spoken with the NPC. The greeting is different for the initial conversation. The second script is used on 3 NPC nodes and checks if the PC has completed the *Beggar's Nest* quest.

The other 5 **When** scripts are attached to 21 PC nodes. Of these 21 PC nodes, 8 have the "nw_d2_intl" script attached to check if the PC has a low intelligence ability score and another 8 PC nodes have the "nw_d2_intn" script attached to check if the PC has a normal intelligence ability score. Another 2 unique **When** scripts combine either the normal or low checks for intelligence with a condition checking whether the PC has completed the *Beggar's Nest* quest. The final unique **When** script checks to see if the PC has a normal intelligence ability score *AND* a high wisdom ability score.

Additionally, the Helmite conversation has 3 unique **What** scripts attached to 5 nodes. The first unique script records the number of times the PC has conversed with the NPC. This number is used in a **When** script to determine if the PC has previously conversed with the NPC. The second unique script makes the NPC perform a series of actions to cast a blessing on the PC. The final unique script opens the NPC's store interface and ends the conversation.

The Helmite conversation built with the dialogue pattern model is shown Figure 6.17. This conversation has 5 decision pattern instances and 24 optional choice pattern instances. The 5 decision instances are instantiated from the 3 decision patterns: *Progress*, *Quest point*, and *Ability*. The 24 optional choice pattern instances are instantiated from the 4 optional choice patterns: *Low Intelligence*, *High Intelligence*, *Ability* and *Quest point*. These patterns are described in Appendix A.

There is one instance of the *Progress* decision pattern. The *First Time* instance ensures that the NPC's introductory greeting remark is only displayed the first time the PC converses with the NPC. Since this pattern instance only affects the very first NPC remark, the pattern instance is a degenerate decision inside the "Welcome" topic. The **Goal** global option is set to the NPC remark under the "First" outcome.

There is one instance of the *Ability* decision pattern. The *Intelligence* instance is a degenerate decision inside the "Goodbye" topic. This decision instance decides on a goodbye remark based on whether the PC has a normal or low intelligence score. It has been adapted by removing the middle outcome. The global **Ability** option is set to *intelligence* and the **Threshold** option for the "High" outcome is set to 9.

[2]Helm is a deity in the Neverwinter Nights setting. Worshipers of this deity are called helmites.

There are three identical instances of the *Quest point* decision pattern. Each instance selects an outcome based on whether the PC completed the *Beggar's Nest* quest. For each instance, the **Quest Point** option for the the "Closed" outcome is set to the final quest point in the *Beggar's Nest* quest. The **Point State** option is set to *Reached.* These pattern instances change the NPC's conversation after the PC completes the quest.

There are 8 instances of the *High Intelligence* optional choice pattern and 7 instances of the *Low Intelligence* optional choice pattern. These instances are attached to choice singletons. The choice singletons cannot be grouped into intelligence choice groups since the NPC responds differently depending on the PC's intelligence. Therefore, a *Low Intelligence* choice singleton links to a different target than the corresponding *High Intelligence* choice singleton. The eighth *High Intelligence* pattern instance is attached to the "[Insight] Is this the only haven?" choice in the topic group that contains the "A yuan-ti", "I know little", "Outside is safe", and "Definitely" topics. This choice does not have a corresponding low intelligence choice.

There is also one instance of the *Ability* optional choice pattern. This instance is also attached to the "[Insight] Is this the only haven?" choice. The **Ability** option is set to *wisdom*, the **Threshold** option is set to *14*, and the **Comparison** option is set to > (greater than). Therefore, this choice is only available if the PC has normal or better intelligence ability score *AND* has high wisdom ability score.

Finally, there are 8 identical instances of the *Quest point* option choice pattern. These pattern instances make a choice available only if the *Beggar's Nest* quest is not completed. Therefore the **Quest Point** option for these instances is set to the final quest point in the *Beggar's Nest* quest and the **Point State** is set to *Not Reached.*

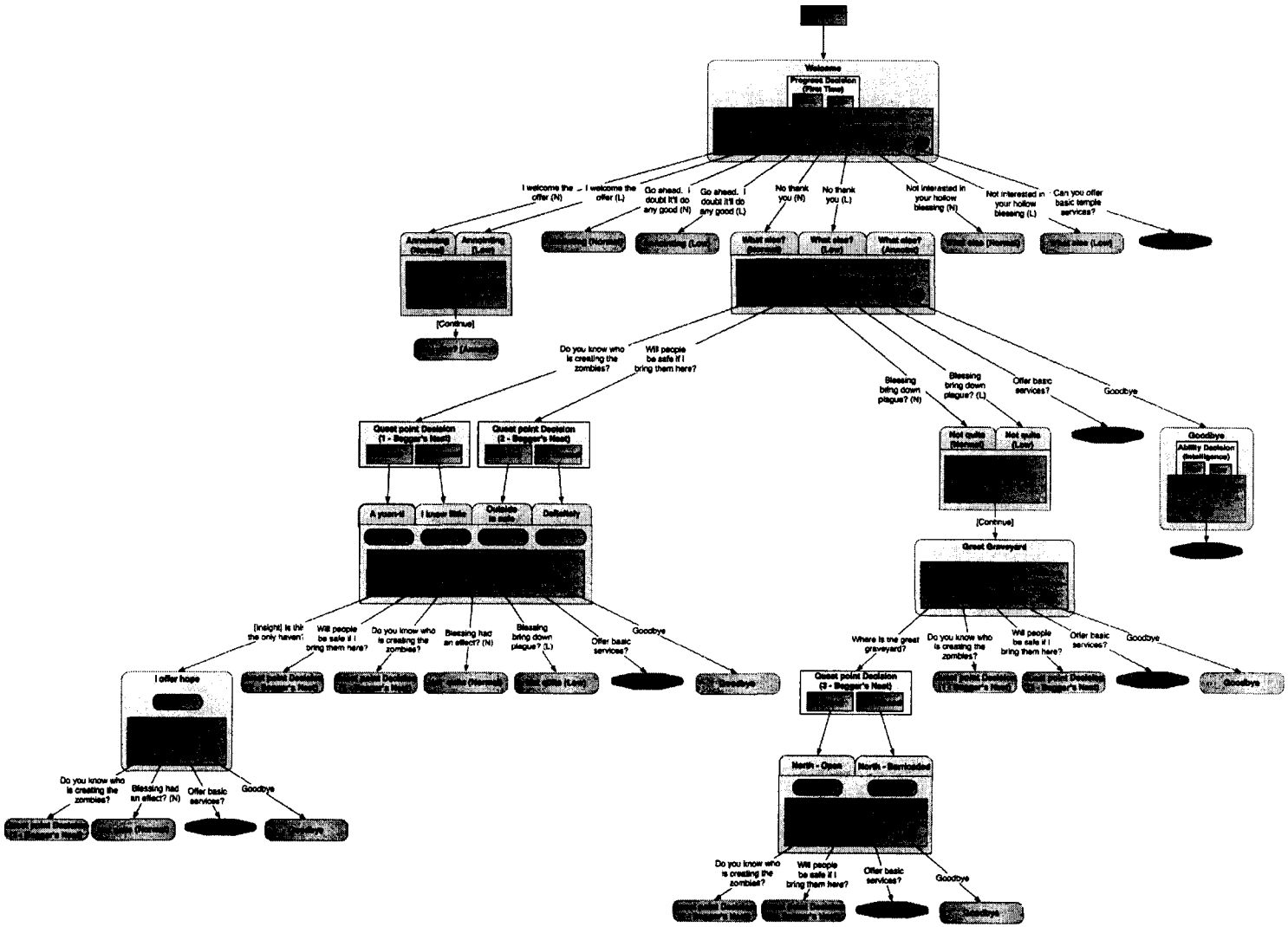


Figure 6.17: The Helmite conversation in the dialogue pattern model (Part 1).

6.2.4 Bertrand

The Bertrand conversation involves a non-trivial quest where the PC must find 2 items, a staff and a journal, that belong to Bertrand's missing brother. As an Aurora conversation, the conversation checks the progress of the quest with several scripts attached to over 40 conversation nodes. The scripts store local variables on the NPC object (i.e. Bertrand) that remember quest details such as whether the items have been found, whether the PC has shown the items to Bertrand, and which items were returned or sold back to Bertrand. Due to the complexity of the quest, Bertrand's conversation is one of the largest conversations in the chapter and one of the most complex. It is the largest conversation in this case study. Therefore, building the Bertrand conversation with dialogue patterns is a good indication of how dialogue patterns can be use to build the most complex conversations.

The Aurora Bertrand conversation has 30 unique **When** scripts attached to 97 nodes. Out of the 30 scripts, 22 of scripts are attached to 23 NPC nodes. These scripts check various conditions relating to the missing brother quest (*A Lost Soul* quest), the PC's charisma, the PC's persuasion skill, and whether the PC conversed with Bertrand previously. The other 8 **When** scripts are attached to 74 PC nodes. Of these 74 PC nodes, 24 have the "nw_d2_intn" script attached to check if the PC has a normal intelligence ability score. Another 24 PC nodes have the "nw_d2_intl" script attached to check if the PC has a low intelligence ability score. The other 6 unique **When** scripts for PC nodes combine either the normal or low conditions for intelligence with a condition checking whether the PC has one or both of the quest items.

Additionally, the Bertrand conversation also has 10 unique **What** scripts attached to 25 nodes. Of these 10 scripts, 6 are attached to 11 NPC nodes and the remaining 4 scripts are attached to 14 PC nodes. These scripts store state to remember what actions the PC has done in regards to the quest, such as returning an item, selling back an item, revealing an item, and lying about the possession of the quest items. Some of these scripts also give the PC experience and gold for returning the items.

The Bertrand conversation built with the dialogue pattern model is shown in 3 different parts in Figures 6.18, 6.19, and 6.20. The red arrow symbols indicate where the parts join to form the entire conversation.

Part 1

Part 1 of the Bertrand conversation (Figure 6.18) has 6 decision pattern instances. Two of them are degenerate decisions. There are no optional choice pattern instances in Part 1. The 6 decision pattern instances are instantiated from 4 decision patterns: *Recall*, *Progress*, *Quest point*, and *Ability* decision patterns. These 4 decision patterns are described in Appendix A.

There are 2 instances of the *Quest point* decision pattern. For the *Items returned* instance, the **Quest Point** option for the "Reached" outcome is set to the "Both Items are Given to Bertrand" quest point in the *A Lost Soul* quest (Finding Bertrand's brother). For the *Marcus dead* instance, the **Quest Point** option for the "Reached" outcome is set to the "Convince Bertrand Marcus is dead"

quest point. For both instances, the "Reached" outcome's **Point State** option is set to *Reached.*

There is one instance of the *Progress* decision pattern. The *First Time* instance ensures that Bertrand introduces himself only once to the PC. The **Goal** option for the "First" outcome is set to the 3 NPC remarks in the "Introduction" topic. The option must include 3 remarks since the "Introduction" topic includes a degenerate decision instance on the first exchange. The degenerate decision has 3 outcomes, therefore there are 3 NPC remarks.

There is one instance of the *Recall* decision pattern. The *Which item is sold* instance recalls which item was sold: the journal, the staff, or neither. To support these three possibilities, the instance is adapted to include a third "Book sold" outcome placed just before the default outcome. The **Point of Interest** global option for the instances is set to 4 PC remarks. The first and second PC remarks are from the choice group labeled "That'll do nicely" in the "No choice - Journal" topic in Part 2 (Figure 6.19). The third and fourth PC remarks are from the choice group labeled "That'll do nicely" in the the "No choice - Staff" topic in Part 2. Although the *Recall* decision pattern eliminates the need for manually creating state variables on NPCs or PCs, each pattern instance still needs to be adapted to recall the appropriate information. The set of ScriptEase actions attached to these 4 remarks is adapted to store "STAFF" if the PC is selling the staff, and "JOURNAL" if the PC is selling the journal. The **Value** option for the "Staff sold" outcome is set to "STAFF", and the **Value** option for the "Book sold" outcome is set to "JOURNAL".

Finally, there are 2 instances of the *Ability* decision pattern. Both are degenerate decision instances. The first instance is in the "Introduction" topic and the second instance is in the "A welcome sight" topic. The global **Ability** option for both instances is set to *charisma.* The **Threshold** options for the "High" and "Medium" outcomes are set to 14 and 9, respectively.

Part 2

Part 2 of the Bertrand conversation (Figure 6.19) has 3 decision pattern instances and 11 optional choice pattern instances. The 3 decision pattern instances are instantiated from the *Skill* decision pattern. The *Skill* decision is described in more detail in Appendix A.

In Figure 6.19, the 3 decision pattern instances are labeled *Persuade - PC's Word, Persuade - Journal,* and *Persuade - Found Nothing.* For each instance, the **Skill** global option is set to *persuasion* since the PC is attempting to persuade Bertrand to believe certain facts and the decision decides whether the PC is successful. For the *Persuade - PC's Word* instance, the **Difficulty** global option is set to *medium.* In this case, the PC is attempting to persuade Bertrand that his brother is dead without any proof. For the *Persuade - Journal* instance, the **Difficulty** global option is set to *easy* since in this case, the PC has some proof that Bertrand's brother is dead (his journal). Finally, for the *Persuade - Found Nothing* instance, the **Difficulty** global option is set to *hard.* In this case, the PC is lying to Bertrand about not finding his brother's possessions. For all 3 decisions, no adaptations were made.

There are 4 optional choice instances attached to 2 choices in the "Brother's name is Marcus" topic in Figure 6.19. The first choice labeled "Marcus? He was a mage?" has 2 optional choice pattern instances attached, *Has item* and *Normal intelligence*, both of which are described in Appendix A. For the *Has item* instance, the **Item** option is set to the *Marcus' Staff* item. The pattern instance is further adapted by adding a definition that determines whether the PC possesses the *Marcus' Journal* item. The ScriptEase condition is modified to be positive if either item is in the PC's inventory. Therefore, this choice is available only if the PC has a normal intelligence ability score *AND* possesses either Marcus' staff or journal.

The second choice labeled "Marcus? Was he spell chucker?" has the same adapted *Has item* optional choice pattern instance. Additionally, it has a *Low intelligence* optional choice instance. Therefore, this choice is available only if the PC has a low intelligence ability score *AND* possesses either Marcus' staff or journal. Since Bertrand replies differently depending on the PC's intelligence, these 2 choices link to two different topics and therefore cannot be in a choice group.

The topic group that includes the "Unfortunate (polite)" topic has 3 optional choices. Each choice has the same adapted *Has item* optional choice instance as the 2 previous optional choices. The first choice singleton, "I'm sorry, but he fell to zombies", also has a *Normal intelligence* optional choice instance. Therefore, this choice is available only if the PC has a normal intelligence ability score *AND* possesses either Marcus' staff or journal. The second choice singleton, "Me not want say this...", has a *Low intelligence* optional choice instance and is available only if the PC has a low intelligence ability score. The third choice is a choice group and consequently only has the adapted *Has item* optional choice attached.

The topic group that includes the "Are you sure? (polite)" topic has 2 optional choices. The first choice, "I found his journal" is a choice group that has a single *Has item* optional choice instance with the **Item** option set to the *Marcus' Journal* item. This pattern instance is not adapted. Therefore this choice is only available if the PC possesses the journal. The second choice, "I found his staff" is also a choice group that has a single *Has item* optional choice instance. In this case, the **Item** option is set to the *Marcus' Staff* item object. Therefore this choice is only available if the PC possesses the staff. The pattern instance is not adapted.

Part 3

Part 3 of the Bertrand conversation (Figure 6.20) has 10 decision pattern instances and 8 optional choice pattern instances. The 10 decision pattern instances are instantiated from the 4 decision patterns: *Recall*, *Has item*, *Quest point*, and *Skill*. These patterns are described in Appendix A.

There are 6 instances of the *Recall* decision pattern. The *Staff sold* instance determines whether the staff was sold or given when the PC returned the staff to Bertrand. The **Point of Interest** global option is set to the 2 PC remarks in the "That'll do nicely." choice group in the "No choice - Staff" topic. These 2 PC remarks are the point where the PC successfully sells the staff to Bertrand. The

Staff sold instance adapts the set of ScriptEase actions attached to these 2 PC remarks to store a string labeled "STAFF". The **Value** option for the instance's "Staff sold" outcome is set to "STAFF".

The *Journal sold* pattern instance is similar. Instead of recalling whether the staff was sold, this pattern instance recalls whether the journal was sold. The **Point of Interest** global option is set to the 2 PC remarks in the "That'll do nicely." choice group in the "No choice - Journal" topic. The *Journal sold* pattern instance adapts the set of ScriptEase actions attached to these 2 PC remarks to store a string labeled "JOURNAL". The **Value** option for the instance's "Book sold" outcome is set to "JOURNAL".

The *Extortion - Which item was sold* and *One was sold* pattern instances are identical (*Recall* decision pattern). Both of these instances recall which item was sold: the journal, the staff, or neither. To support these three possibilities, both instances are adapted to include a third "Book sold" outcome that is placed just before the default outcome. The **Point of Interest** global option for both instances is set to the 4 PC remarks involved in both the **Staff sold** and **Journal sold** *Recall* decision instances. The ScriptEase actions for those 4 remarks are adapted to store "STAFF" if the PC is selling the staff, and "JOURNAL" if the PC is selling the journal. The **Value** option for the "Staff sold" outcome is set to "STAFF", and the **Value** option for the "Book sold" outcome is set to "JOURNAL".

The final two *Recall* decision instances, "1 - Most recent shown" and "2 - Most recent shown", are also identical. Both of these instances recall which items were revealed to Bertrand during the conversation: the journal, the staff, or neither. To support the three possibilities, both instances are adapted by adding a third "Journal shown" outcome before the default outcome. The **Point of Interest** global option for both instances includes 1 NPC remark and 2 PC remarks. The NPC remark is from the first exchange in the "I believe you (Journal)" topic in Part 2 (Figure 6.19). The 2 PC remarks are from the "I found his staff" choice group in the topic group that includes the "Are you sure? (polite)" topic. The ScriptEase actions for these 3 remarks are adapted to store "STAFF" if the staff has been shown, and "JOURNAL" if the journal has been shown. The **Value** option for the "Staff shown" outcome is set to "STAFF", and the **Value** option for the "Journal shown" outcome is set to "JOURNAL".

There are 2 identical instances of the *Quest point* decision pattern. Both the *Extortion* and *Either item returned* pattern instances use the *A Lost Soul* (Finding Bertrand's brother) quest to determine which items have been returned (i.e. either sold or given) to Bertrand. Both instances are adapted to include a third "Journal returned" outcome to check if the journal has been returned. The condition for the "Journal returned" outcome is the same as the "Staff returned" outcome. Both the "Staff returned" and "Journal returned" outcomes have their **Point State** options set to *Reached.* The **Quest point** option for the "Staff returned" outcome is set to the "Return Staff to Bertrand" quest point. The **Quest Point** option for the "Journal returned" outcome is set to the "Return Journal to Bertrand" quest point.

There is one instance of the *Has item* decision pattern. The *Staff OR Journal* pattern instance is adapted by adding a third "Has Journal" outcome that determines if the PC possess the journal. The added outcome shares the same condition and options as the "Has Staff" outcome. The **Item** option for the "Has Staff" outcome is set to *Marcus' Staff.* The **Item** option for the "Has Journal" outcome is set to *Marcus' Journal.*

Finally, there is one instance of the *Skill* decision pattern. The *Persuade - Nothing Else* instance is used when the PC attempts to lie to Bertrand about finding other items on Marcus' body. For the "Success" outcome, the **Skill** option is set to *persuasion* and the **Difficulty** option is set to *Hard.*

Additionally, part 3 of the Bertrand conversation has 8 instances of the *Has item* optional choice pattern. The first 2 instances are attached to the two choice groups in the topic group containing the "I believe you (Journal)" topic. The **Item** option for both instances is set to *Marcus' Staff.* These instances are also adapted to include extra definitions that check if the item *Marcus' Journal* is in the PC's inventory. The ScriptEase condition is adapted to be positive if the PC contains either the staff or the journal. Therefore these choices are available only if either the staff or the journal are in the PC's inventory.

Another 3 *Has item* pattern instances have the **Item** option set to the *Marcus' Journal* item. One of these 3 instances is attached to the "He had a journal. Take it." choice group in the "Cannot believe..." topic. The other 2 pattern instances are attached to the "I have his journal. I'll give it to you" and "I demand payment for journal." choices in the topic group containing the "Staff sold" topic.

The final 3 *Has item* pattern instances have the **Item** option set to the *Marcus' Staff* item. One of these 3 instances is attached to the "He had a staff. Take it." choice group in the "Cannot believe..." topic. The other 2 pattern instances are attached to the "I have his staff. I'll give it to you" and "I demand payment for staff." choices in the topic group containing the "Journal sold" topic.

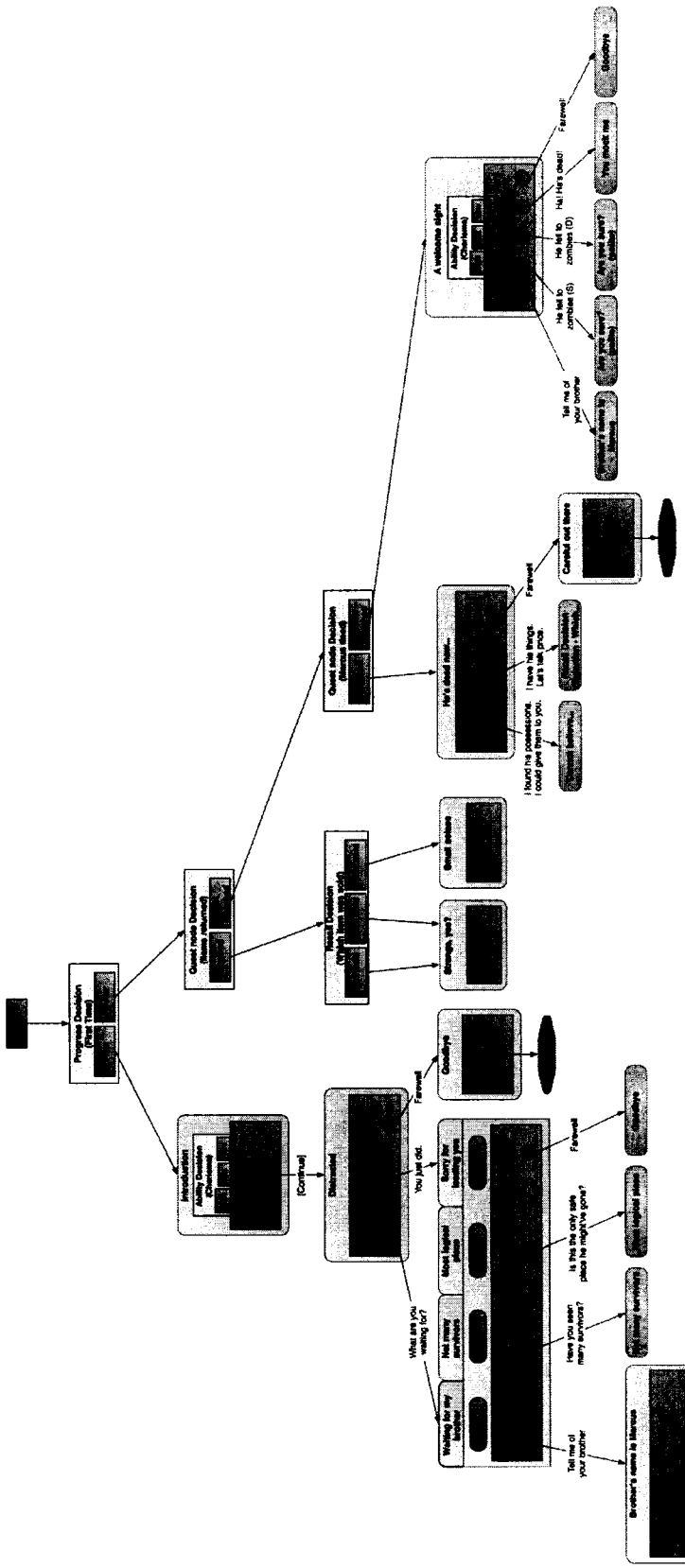


Figure 6.18: The Bertrand conversation in the dialogue pattern model (Part 1).

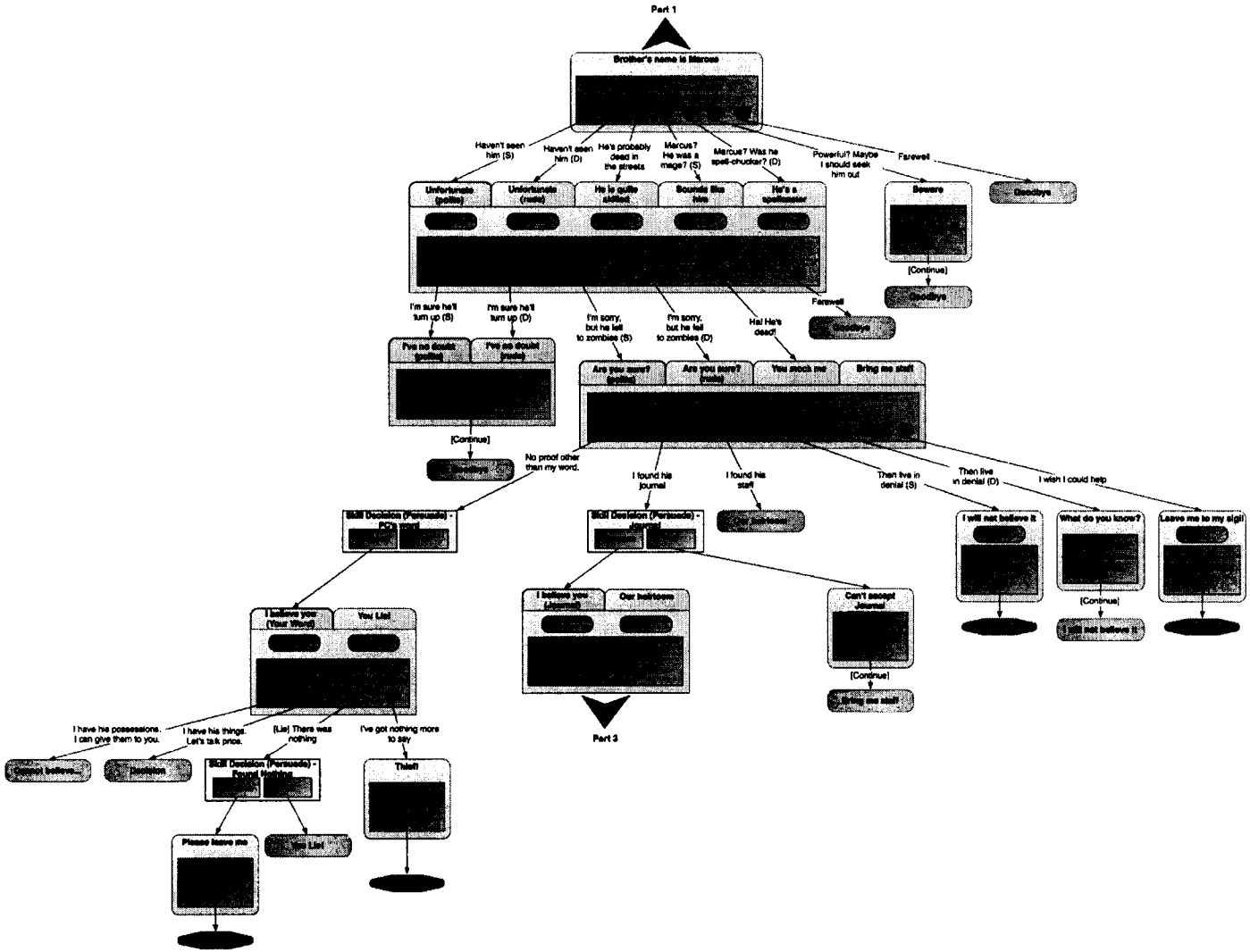


Figure 6.19: The Bertrand conversation in the dialogue pattern model (Part 2).

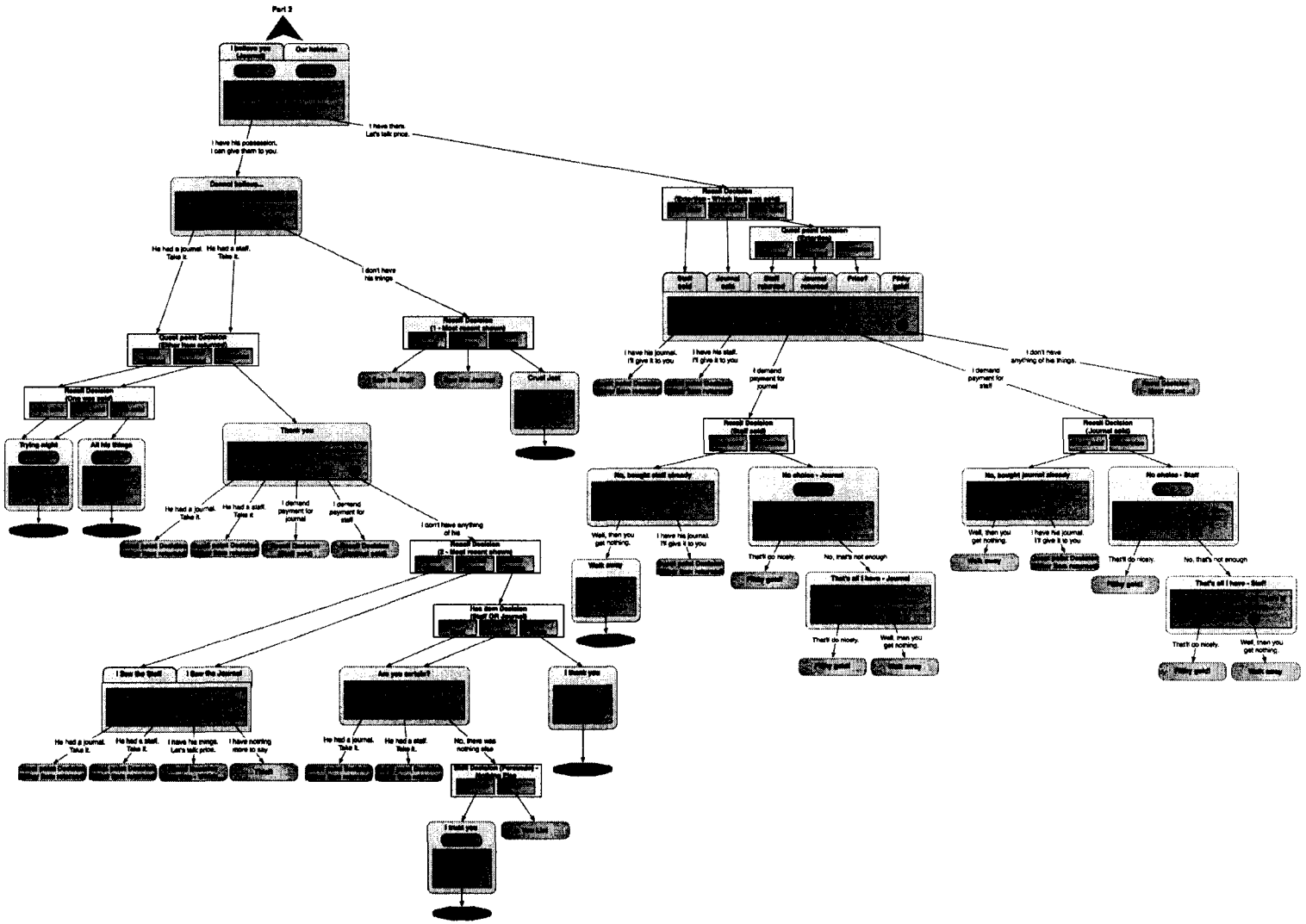


Figure 6.20: The Bertrand conversation in the dialogue pattern model (Part 3).

6.3 Results

This section will use the metrics presented in Section 6.1 with the conversations presented in Section 6.2 to compute and compare the complexities of the Aurora conversation model and the dialogue pattern model. For the Aurora conversation model, the number of nodes for each conversation is shown in Table 6.2 and the operation count for each conversation is shown in Table 6.3. For the dialogue pattern model, the number of components for each conversation is shown in Table 6.4 and the operation count for each conversation is shown in Table 6.5. The numbers in these tables are used to compute the complexity metrics.

Node	Emernick	Nurse	Helmite	Bertrand
NPC	25	38	26	77
PC	39	49	31	97
Link	89	125	94	267
Total	176	189	151	441

Table 6.2: Node counts for the conversations under the Aurora conversation model.

Operation	Emernick	Nurse	Helmite	Bertrand
Add Node	64	87	57	174
Edit Text	64	87	57	174
Add Link	125	89	94	267
Write Script	13	8	10	40
Attach Script	49	38	30	122
Total	315	309	248	777

Table 6.3: Operation counts for the conversations under the Aurora conversation model.

Component	Emernick	Nurse	Helmite	Bertrand
Topics	25	22	17	58
Tail Exchanges	25	28	21	62
Inner Exchanges	0	10	7	12
Inner Choice Singletons	0	8	7	12
Inner Choice Groups	0	2	0	0
Tail Choice Singletons	3	22	21	27
Tail Choice Groups	33	11	17	59
Secondary Links	24	18	19	45
Decision Patterns	4	1	3	17
Degenerate Decision Patterns	0	3	2	2
Optional Choice Patterns	1	10	24	15
External Encounter Patterns	5	0	4	19
Total	120	135	142	328

Table 6.4: Component counts for the conversations under the dialogue pattern model.

The complexity metric results are shown in Table 6.6. Each conversation has two columns, one for the Aurora conversation model (**Aurora**) and the other for the dialogue pattern model (**D. Patts**). A metric score is in **bold** if it is the lowest score.

Operation	Emernick	Nurse	Helmite	Bertrand
Add Topic	25	22	17	58
Merge Topic	4	5	5	8
Insert Exchange	0	10	7	12
Add Decision Pattern	4	1	3	17
Add Degenerate Decision Pattern	0	3	2	2
Add Choice Singleton	0	13	15	10
Remove Choice Singleton	3	0	0	2
Convert Choice to Group - *Intelligence*	7	5	3	18
Add Choice Group - *Intelligence*	26	8	14	41
Link	24	18	19	50
Set Remark Text	94	98	90	231
Add Optional Choice Pattern	1	1	24	15
Add *What* Encounter Pattern	5	0	4	19
Total	193	184	203	471

Table 6.5: Operation counts for the conversations under the dialogue pattern model.

Complexity	Emernick		Nurse		Helmite		Bertrand	
	Aurora	D. Patts	Aurora	D. Patts	Aurora	D. Patts	Aurora	D. Patts
Component	176	**120**	189	**135**	151	**142**	441	**328**
Reduced Component	-	85	-	99	-	92	-	217
Structural	176	**120**	189	**115**	151	**128**	441	**304**
Reduced Structural	-	85	-	79	-	78	-	193
Remark	**64**	94	*87*	94	**57**	90	**174**	231
Indirection	125	**24**	89	**18**	94	**19**	267	**45**
Operational	309	**193**	315	**184**	248	**204**	777	**471**

Table 6.6: Complexity results for the four conversations.

The dialogue pattern model scores better than the Aurora conversation model in both the component and structural complexity scores for all four conversations. Even though the value-added components make up 30% of the *Emernick* conversation, 26% of the *Nurse* conversation, 35% of the *Helmite* conversation, and 34% of the *Bertrand* conversation, the dialogue pattern model has a lower overall number of components. This can be attributed to a large reduction in the number of redirections, the grouping of choices into choice groups, and the sharing of choices inside topic groups. The small difference in component and structural complexity scores show that these four conversations do not have many inner exchanges. Even the *Bertrand* conversation, with 58 topics, only has 12 inner exchanges. Collapsing topics to hide inner exchanges only has a small effect on the number of visible components, though it is still helpful.

The reduction of indirections and the introduction of choices groups and topic groups also counteract the extra duplication of choices in the dialogue pattern model. As an example of this extra duplication, consider the "Goodbye" topic in the Helmite conversation (Figure 6.17). It is directly linked by a choice grouped labeled "Goodbye". However, it is also linked via secondary links by several other "Goodbye" choice groups found in "A yuan-ti", "I offer hope", "Great Graveyard", and "North - Open" topics. In the Aurora conversation model, these would be represented as link

nodes that link to the original "Goodbye" PC node. However, in the dialogue pattern model these are duplicate choice groups that link to the "Goodbye" topic. In the Aurora conversation model, the link nodes are counted once each, but in this case each duplicated choice effectively counts as two, one for the duplicated choice, and one for the secondary link back to the "Goodbye" topic.

For all four conversations, the dialogue pattern model has a much better indirection complexity score than the Aurora conversation model. In fact the dialogue pattern model complexity score is 80-83% lower than its Aurora conversation model counterpart for all conversations. These vastly lower scores can be attributed to the sharing of choices inside topic groups. For example, in Part 1 of the Nurse conversation (Figure 6.15) there is a topic group that contains the "Plague", "Any more questions?", "Cure", and "I was a midwife" topics. There are five choices (four choice groups and one choice singleton) shared among these four topics. In the Aurora conversation model, these topics are represented as one or more NPC nodes. The NPC nodes for the "Plague" topic contain 9 PC nodes that represent the five choices. The NPC nodes of the other three topics would each contain 9 link nodes that link to these 9 PC nodes for a total of 27 indirections in the section of conversation. The dialogue pattern model eliminates these 27 indirections since the choices are shared inside the topic group. Instead, there are only four indirections which are secondary links to target topics. This is an 85% reduction in the number of indirections which is close to the reduction percentage for the overall indirection complexity score. Other topic groups have a similar reduction. This is an enormous benefit to the author since this makes conversations built with dialogue patterns are much less disjointed and therefore easier to navigate.

In all four conversations, the operational complexity of the conversation built with dialogue patterns is noticeably lower than the Aurora conversation model counterpart. The dialogue pattern model uses fewer operations, partially due to the fact that the **Add Topic** creates a new topic, exchange, and choice all at once. The sharing of choices between topics in a topic group also reduces the number of operations needed. For example, consider in Part 1 of the Nurse conversation (Figure 6.15). There is a topic group mentioned previously that contains the "Plague", "Any more questions?", "Cure", and "I was a midwife" topics. In the Aurora conversation model, the equivalent piece of conversation is built with 29 **Add Node** operations since there are 9 NPC nodes and 20 PC nodes. There are also 29 **Edit Text** operations, one for each node. Additionally there are 18 **Add Link** operations so that each "topic" of NPC nodes share the same 9 PC nodes described earlier. Finally there are 12 **Attach Script** operations to attach the normal and low intelligence scripts to 12 PC nodes. Thus, the author must perform 88 operations to construct this piece of conversation in the Aurora conversation model.

Constructing the same topic group in the dialogue pattern model requires four **Add Topic** and one **Merge Topic** operations. Recall that for counting purposes, topic groups are constructed by adding the first two topics of the group to the conversation with two **Add Topic** operations, merging them with one **Merge Topic** operation, then adding the remaining topics to the topic group with the

Add Topic operation. The inner exchanges are constructed using 7 **Insert Exchange** operations. Of these 7 inner exchanges, two inner exchanges have their choice singleton changed to a choice group using the **Convert Choice to Group - *Intelligence*** operation. At this point, the four topics only share a single choice singleton. This singleton is converted to a choice group with one **Convert Choice to Group - *Intelligence*** operation. Then the other four choices are added with one **Add Choice Singleton** and three **Add Choice Group - *Intelligence*** operations. Finally, the remark text is set for all NPC and PC remarks using 29 **Set Remark Text** operations. The author does not need to instantiate any optional choice patterns since these pattern instances are automatically created with the **Convert Choice to Group - *Intelligence*** and **Add Choice Group - *Intelligence*** operations. Thus, the author must perform 48 operations to construct this piece of conversation in the dialogue pattern model. Therefore 40 fewer operations are used to construct this piece of conversation in the dialogue pattern model compared the equivalent piece of conversation in the Aurora conversation model.

The lower operational complexity scores result from using a larger set of operations that construct several components simultaneously rather than a small set of atomic operations that construct only single components. It could be argued that a larger set of operations is more confusing to the author. However, in a GUI these operations can be context-sensitive and only be available if the operation can be performed legally. Therefore the author only sees a small subset of the possible set of operation at one time.

The duplication of choices in the dialogue pattern model has a significant impact in the remark complexity scores. For each conversation, the dialogue pattern model scores much worse than its Aurora conversation model counterpart. For example, in the case of the "Goodbye" choice in the *Helmite* conversation (Figure 6.17), the Aurora conversation model has only two PC nodes, one for normal intelligence and one for low intelligence, with the remaining "Goodbye" nodes as link nodes. Therefore the author only sets the remark text for two PC nodes. For the dialogue pattern model, the author must set the text for 8 PC remarks since the "Goodbye" choice is duplicated in four topics other than the original topic, even though the same two PC remarks are repeated in the other four pairs. The duplication of choices also affects the *Emernick*, *Bertrand*, and to a lesser degree the *Nurse* conversations. The difference in remark complexity scores for the *Nurse* conversation is small since the conversation only has four duplicated "Goodbye" choice singletons that link to the "Goodbye" topic and three duplicated "[Continue]" choice singletons that link to the "More Gossip?" topic.

The remark complexity reveals a disadvantage in the dialogue pattern model. The author must set the text for more PC remarks than an equivalent conversation in the Aurora conversation model. Some of the burden can be mitigated by copying and pasting text. However, if the author changes the remark text for one choice, all duplicate choices that share the same remark text must also be changed. Fortunately, this problem can be further mitigated by a GUI which assists the author

with changing remark text of duplicate choices. The GUI can track which choices have identical remark text that link to the same target. If the author changes one of these choices, the GUI can automatically change the text of the other choices with or without author confirmation.

6.4 Summary

This chapter introduced a case study of four conversations from Chapter 1 of the Neverwinter Nights official campaign. A set of five complexity metrics were introduced to directly measure the effectiveness of the dialogue pattern model against the Aurora conversation model. This comparison was independent of the quality of tools implementing the models. The *Emernick*, *Nurse*, *Helmite*, and *Bertrand* conversations were then described, including the identification of the decision and optional choice patterns used in each conversation. Finally, the conversations from both models were compared using the five complexity measures. The dialogue pattern model has better **component** and **structural** complexity scores than the Aurora conversation model, despite having extra value-added components. The dialogue pattern model had a better **indirection** complexity since topic groups and choice groups reduced the number of secondary links. This may be the most important complexity measure since changing focus involves a context switch that can be disruptive to the author. Additionally, the dialogue pattern model had better **operational** complexity since fewer operations are required to construct a topic group than the equivalent set of conversation nodes in the Aurora conversation model. Conversations under the dialogue pattern model has worse **remark** complexity since there was duplication of choices between topics not in topic groups. This duplication created extra PC remarks for the author to populate with text. This drawback can be mostly mitigated by a good GUI that helps the author manage duplicated PC remarks.

It has been demonstrated that the dialogue pattern model provides an improvement over the Aurora conversation model. The new model abstracts conversations so that they can be presented more compactly. This allows the author to find the sections of conversation that need attention. It also allows the author to construct more complex conversations in a shorter period of time. Decision and optional choice patterns save the author from writing scripting code to make the conversations more dynamic and flexible. Perhaps most importantly, fewer distracting context changes are necessary. This case study provides enough evidence of the superiority of the dialogue pattern model that is should be implemented and tested with a user-study.

Chapter 7

Future Work and Conclusion

7.1 Future Work

This dissertation presents a new model for building conversations using dialogue patterns. However, at the time of writing there is no tool that implements this new model. The next logical step is to build a conversation tool that integrates with the existing ScriptEase application. After this tool is built, user studies could be performed to measure the ease of use of building conversation with dialogue patterns. Feedback from these studies could be used to improve both the implemented tool and the model itself. Additionally, it would be useful to measure the effectiveness of using patterns rather than scripts by recording how long it would take for a single author to rebuild the conversations in Chapter 1. A more comprehensive study would combine dialogue patterns with plot, behaviour, and encounter patterns to replace every single script in the module.

Although this research allows the creation of more detailed conversations in a shorter amount of time, the author must still create the prose or text for every single remark in each conversation. It would be useful to find ways of automatically generating some of this conversation using NLP techniques. For example, the author may want to create a conversation for an NPC that can assign the PC a *Retrieve an item* quest. The conversation would have a common structure, such as a remark that assigns a quest, a PC choice that gives the item to the NPC once the PC has retrieved it, and decision and optional choice patterns that guard sections of the conversation depending on the state of the quest. These could be represented as a basic *Retrieve an Item Quest* dialogue pattern. However, the author would still need to enter remark text for all remarks. Using NLP techniques, it would be helpful to automatically enter the text for these remarks using the context of the virtual world and the options already set in the *Retrieve an item* quest instance. Ultimately the author could choose from conversation templates that do much of the dialogue generation automatically. The author could then focus time on tuning the conversation instead of creating it.

7.2 Conclusion

In Chapter 1, this dissertation introduced the computer role-playing game genre with Neverwinter Nights as the primary example. The Aurora toolset was introduced to demonstrate how modern CRPGs are built. The toolset uses a CAD-like interface combined with hand-written scripts to build a dynamic world and story. This dissertation focused on the Aurora conversation editor and its disadvantages. In Chapter 2, the application of design patterns in the form of ScriptEase was shown to be a better alternative than manual scripting to build interactivity in the domain of CRPGs. The ScriptEase tool was described with an example of building an encounter pattern.

In Chapter 3, the structural components of the dialogue pattern model were introduced as an alternative to the model used by the Aurora conversation editor. Exchanges, topics, topic groups, choices, and primary and secondary links were explained. In Chapter 4, the model was further expanded with the introduction of decision patterns and optional choice patterns. These patterns replace the manual scripting used by the Aurora conversation model with basic ScriptEase components. The structural components in Chapter 3 were combined with these pattern in Chapter 4 to form dialogue patterns. In Chapter 5, the operations used to construct the components for dialogue patterns were described.

Finally, in Chapter 6 the dialogue pattern model was evaluated directly against the Aurora conversation model using complexity metrics. The *component*, *structural*, *remark*, *indirection*, and *operational* complexity metrics were computed for four conversations from chapter 1 of the Neverwinter Nights official campaign. The dialogue pattern model was shown to be an improvement over the Aurora conversation model. However, it was revealed that the drawback of the dialogue pattern model was having extra duplicated choices and therefore requiring the author to set the text for more remarks than the equivalent conversation in the Aurora conversation model. It was suggested that this problem can be mitigated with support from the GUI that implements the model. This dissertation has provided evidence for the efficacy of the dialogue pattern model. A ScriptEase implementation of this model should do for conversation authoring what encounter patterns have done for authoring PC-object interactivity.

Bibliography

[1] Bioware Corp.
http://www.bioware.com.

[2] M. Cutumisu, C. Onuczko, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, J. Siegel, and M. Carbonaro. Evaluating Pattern Catalogs - The Computer Games Experience. In *Proceedings of the 28th International Conference on Software Engineering*, pages 132–141, May 2006.

[3] M. Cutumisu, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko, and M. Carbonaro. Generating Ambient Behaviors in Computer Role-Playing Games. *IEEE Intelligent Systems*, 21(5):88–99, 2006.

[4] The D20 Game Engine.
http://www.wizards.com/default.asp?x=d20/welcome/.

[5] Packard E. Choose Your Own Adventure series, 1979-89.

[6] Ron Edwards. *GNS and Other Matters of Role-playing Theory*. Adept Press, 2001.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Resuable Object-Oriented Software*. Addison-Wesley Professional, September 1994.

[8] L. Henry. Group Narration: Power, Information, and Play in Role Playing Games.
http://www.darkshire.net/ jhkim/rpg/theory/liz-paper-2003/, May 2003.

[9] Jade Empire.
http://jade.bioware.com.

[10] G. Kacmarcik. Question-Answering in Role-Playing Games. In *Papers from the AAAI Workshop on Question Answering in Restricted Domains*, pages 51–55. AAAI Press, 2005.

[11] G. Kacmarcik. Using Natural Language to Manage NPC Dialog. In *Artificial Intelligence and Interactive Digital Entertainment*, pages 115–117, 2006.

[12] Knights of the Old Republic.
http://www.bioware.com/games/knights_old_republic/.

[13] Lilac Soul's Script Generator.
http://nwvault.ign.com/View.php?view=Other.Detail&id=625.

[14] M. Mateas and A. Stern. Facade: An Experiment in Building a Fully-Realized Interactive Drama. In *Game Developers Conference*, March 2003.

[15] M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker. ScriptEase: Generative Design Patterns for Computer Role-Playing Games. In *Proceedings of the 19th IEEE Conference on Automated Software Engineering (ASE 2004)*, pages 88–99, Linz, Austria, September 2004.

[16] M. McNaughton, J. Schaeffer, D. Szafron, D. Parker, and J. Redford. Code Generation for AI Scripting in Computer Role-Playing Games. In *Challenges in Game AI Workshop at AAAI-04*, pages 129–133, San Jose, USA, July 2004.

[17] Neverwinter Nights Awards.
http://nwn.bioware.com/about/awards.html.

[18] Neverwinter Nights Community Website.
http://nwn.bioware.com.

[19] Oblivion.
http://www.elderscrolls.com/games/oblivion_overview.htm.

[20] K. Tan, D. Szafron, J. Schaeffer, J. Anvik, and S. MacDonald. Using Generative Design Patterns to Generate Parallel Code for a Distributed Memory Environment. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 203–215, 2003.

[21] A. Tychsen, M. Hitchens, T. Brolund, and M. Kavakli. The game master. In *IE2005: Proceedings of the second Australasian conference on Interactive entertainment*, pages 215–222, Sydney, Australia, Australia, 2005. Creativity & Cognition Studios Press.

Appendix A

Dialogue Pattern Catalog

This appendix describes the individual patterns identified for this dissertation. The majority of these patterns are used in the case study conversations described in Section 6.2.

A.1 Decision Patterns

The decision patterns identified in this dissertation are described as follows:

A.1.1 *Ability* Decision

The *Ability* decision decides on whether one of the PC's ability scores, such as *charisma* or *wisdom*, is either high, medium, or low. The **Ability** global option determines the ability in question. Both the "High" and "Medium" outcomes have a **Threshold** option. If the PC's ability score is higher than the **Threshold**, then the outcome is selected. The "High" outcome placed before the "Medium" outcome, otherwise the medium outcome would always be selected first. The "Low" outcome is the default outcome.

A.1.2 *Basic gender* Decision

The *Basic gender* decision decides on the PC's gender. Although the player can only create male and female characters, Neverwinter Nights identifies several different genders for NPCs: female, male, both, and neutral. This decision is useful to differentiate between female and male player characters. The *Basic gender* decision applies to PCs and has "Female" and "Male" outcomes with "Male" as the *default* outcome. Similar to the *Ability* decision pattern, this pattern is designed to decide specifically on a PC characteristic, and therefore requires no additional options to function properly. A more general *Gender* decision provides outcomes for all possible genders, based on any target creature provided as a global option.

A.1.3 *Door locked* Decision

The *Door locked* decision decides on the locked status of a door. A door can either be "Locked" or "Unlocked" and the pattern has an outcome for each state with "Unlocked" as the *default* outcome. The pattern also has a single global option **Door** so that the author can select the target door. Unlike the *Ability* and *Basic gender* decisions, the *Door locked* decision does not use PC characteristics to select an outcome.

A.1.4 *Near by* Decision

The *Near by* decision decides whether a game object is within a certain distance of another game object. The "Inside" outcome is the first outcome and is selected when the two objects are within a certain distance. The "Outside" outcome is the *default* outcome. The pattern as three global options. Both the **First Object** and **Second Object** options can be any game objects. The **Distance** option is a *float* representing the distance in meters.

A.1.5 *Progress* Decision

The *Progress* decision decides on an outcome based on which remarks in the conversation have been previously *visited.* A NPC remark is considered visited if the remark is displayed in conversation. A PC remark is visited if the player selects it as a response to the NPC. For example, a conversation could make an early decision on whether the NPC greets the PC with either "Hello stranger!" or "So we meet again!". The first remark would be selected if it was the first time the PC has conversed with the NPC. The second remark would be selected for all subsequent conversations. This decision selects an outcome based on whether the "Hello stranger" remark has previously visited. If it has been previously visited then the decision will select the "So we meet again!" outcome.

The *Progress* decision has two outcomes. The first outcome labeled "Initial" has one remark option called **Goal**. Similar to other game objects in ScriptEase, the author would select the **Goal** remark from a picker. The **Goal** option specifies which remark needs to be visited in order for the "Initial" outcome to be *not* selected. For example, the author can select the the "Hello stranger!" remark as the **Goal** option. The second outcome "Final" is the *default* outcome. This outcome is always selected after the **Goal** remark has been reached. In this example, the "Initial" outcome links to the topic with the "Hello Stranger" remark which is same as the **Goal** option. This allows the outcome to be selected once and once only which is useful for first-time greetings in conversations.

In a second example, the author may want the PC to ask the NPC for a favour. Using a *Progress* decision, the NPC's reply can either be "Sure, I'll help." for the "Initial" outcome or "You've said enough. Goodbye" for the "Final" outcome. The "Final" outcome is selected if the PC insults the NPC in another part of the conversation by visiting the "You have a face only a mother could love." remark. This is done by setting the **Goal** option for the "Initial" outcome to this insulting remark. Now the NPC would be happy to assist the PC unless the PC decides to insult the NPC.

The adaptations described in Section 4.1.4 give the *Progress* decision extra flexibility. Each new outcome includes its own **Goal** option, allowing the author to create any number of "phases" for a single decision. Intuitively, the decision "progresses" from the first outcome to the final *default* outcome as more goal remarks in the conversation are visited. The pattern can also be generalized by changing the **Goal** option from a remark to a *list* of remarks. In this case, if any one remark in the list is visited during conversation, then the outcome will no longer be selected.

Section 4.1.2 describes that decision patterns generate **When** scripts for NPC conversation nodes that put conditions on an appearance of remarks. However, in order for the *Progress* decision pattern to function, it needs to associate actions with each remark that is specified as an **Goal** option. These actions set a local variable on the NPC to indicate which remarks have been visited. The *Progress* decision pattern uses these local variables to determine which outcome to select.

A.1.6 *Recall* Decision

The *Recall* decision makes a decision based on a small piece of game state that was stored at a certain point in a conversation. The author can use this pattern to make a decision on information that was relevant at an arbitrary point in an arbitrary conversation. For example, the author wants the NPC to greet the PC differently depending on whether the PC lied to the NPC about having a special item earlier in the conversation. The PC has the choice to lie or tell the truth, and that choice is recorded by the decision. When the PC talks to the NPC a second time, the decision can then *recall* the recorded information to decide how the NPC will greet the PC.

The *Recall* decision has a "First" outcome and "Last" default outcome. The pattern has a remark global option called **Point of Interest** which represents the remark in the conversation where the decision needs to remember a piece of game state. The decision pattern remembers information in the form of strings. The "First" outcome has a **Value** string option. This option is compared against the string stored when the **Point of Interest** remark was visited. If the strings match, the outcome is selected. Otherwise the default "Last" outcome is selected. The pattern can be adapted by adding additional outcomes, where each outcome's **Value** string option is compared against the remembered string.

Similar to the *Progress* decision, the *Recall* decision requires a actions to be attached to the **Point of Interest** remark. These actions stores a string as a local variable on the NPC. However, the *Recall* decision differs from the *Progress* decision since the remark option is a global option rather than a outcome option. Also, the pattern decides on game state stored when the **Point of Interest** remark was visited, and not on whether the remark was visited. Consequently, the author needs to specify the piece of game state that the *Recall* pattern uses by adapting the actions that are stored on the **Point of Interest** remark. This process is described in 4.1.5.

A.1.7 *Is the PC* Decision

The *Is the PC* decision decides whether the PC is currently speaking to the NPC. Scripts can trigger the conversation to be started without a PC so that the NPC will speak a one-liner. Instead of a conversation window opening, this one-liner appears above the NPC's head. This decision has two outcomes. The "PC" outcome is selected there is a PC involved in the conversation. The default "Other" outcome is selected if the conversation was started via script without the PC. There are no options for this decision pattern. In most cases, the author connects an *utterance* topic to the "Other" outcome so that the NPC speaks a one-liner.

A.1.8 *Quest point* Decision

The *Quest point* decision decides whether a certain *quest point* has reached a certain state. A quest in Neverwinter Nights can be represented as a plot pattern. A plot pattern represents significant events relating to the quest as quest points. For example, a *Retrieve an Item* quest would have a quest point representing when the player acquires the quest item. Other patterns, including dialogue patterns, can query the state of these quest points. A quest point either be *reached*, or *not reached*. A quest point is reached if the its event has already occurred. For example, when the PC has acquire the item, the quest point is considered *reached*. A quest point is *not reached* if the event has not yet occurred.

This decision has two outcomes. The "Reached" outcome has two options. The **Quest Point** option determines the quest point. This option would be selected from a picker that shows the list of all quests along with their quest points. The **Point State** determines the state of the quest point and can be either *reached* or *not reached*. The "Otherwise" outcome is the default outcome.

A.1.9 *Skill* Decision

The *Skill* decision performs a skill check based on the PC's proficiency of a specific skill. This decision has two outcomes. The "Success" outcome is selected if the skill check is successful. The default "Failure" outcome is selected if the check fails. There are two global options. The **Skill** option determines the skill to be checked. The **Difficulty** option determines the difficulty of the check and be *easy*, *medium*, *hard*, *superior*, *master*, *legendary*, or *epic*. Each level is more difficult than the previous. The difficulty is relative to the PC's character level and therefore scales as the PC gains levels.

A.1.10 *Has item* Decision

The *Has item* decision decides whether a specific item is in the PC's inventory. This decision has two outcomes. The "Has item" outcome has a single **Item** option that determines the item to check. This outcome is selected if the **Item** is in the PC's inventory. The "Otherwise" outcome is the default

outcome. This decision is easily adapted to add additional outcomes. Each outcome checks for a specific item and the "Otherwise" outcome is selected if the PC possesses none of the items.

A.2 Optional Choice Patterns

The optional choice patterns identified in this dissertation are described as follows:

A.2.1 *Ability* Optional Choice

The *Ability* optional choice pattern makes a choice available when one of the PC's ability scores meets a certain condition. The **Ability** option determines the ability score that is compared. The **Comparison** option determines the type of comparison used. It can be $<$, $\leq$, $=$, $\geq$, or $>$. The **Threshold** determines the number against which the ability score is compared. For the conversations in the case study in Chapter 6, this pattern is used frequently to ensure the PC's wisdom score is a high value (14 or more).

A.2.2 *Normal Intelligence* Optional Choice

The *Normal Intelligence* optional choice pattern is a specialization of the *Ability* decision. It makes a choice available only if the PC has a "normal" intelligence ability score greater than 9. The **Ability** option is automatically set to *intelligence*, the **Comparison** option to $>$, and the **Threshold** to 9. This pattern is used in the *Choice group - Intelligence* pattern. It is also frequently used in the conversations in the case study presented in Chapter 6.

A.2.3 *Low Intelligence* Optional Choice

The *Low Intelligence* optional choice pattern is a specialization of the *Ability* decision. It makes a choice available only if the PC has a "low" intelligence ability score of 9 or lower. The **Ability** option is automatically set to *intelligence*, the **Comparison** option to $\leq$, and the **Threshold** to 9. This pattern is used in the *Choice group - Intelligence* pattern. It is also frequently used in the conversations in the case study presented in Chapter 6.

A.2.4 *Has item* Optional Choice

The *Has item* optional choice pattern makes a choice available only if a specified item is in the PC's inventory. The **Item** option specifies the item.

A.2.5 *Quest point* Optional Choice

The *Quest point* optional choice pattern makes a choice available only if a specified quest point is in a specified state. A quest point is described in Section A.1.8. The **Quest Point** option specified the quest point. The **Point State** specifies the desired state of the quest point and can be either *reached* or *not reached*.