Dialogue Patterns - A Visual Language For Dynamic Dialogue

J. Siegel, D. Szafron*

Department of Computing Science, University of Alberta, Edmonton, AB, Canada, T6G 2E8

Received:

Abstract

A dynamic dialogue is a conversation in which each participant alternately selects remarks based on a changing world state and in which each remark can change the world state. Dynamic dialogues happen frequently as conversations between a player character (PC) and a non-player character (NPC) in a computer game. When it is the PC's turn to speak, the current game state is used to filter the static set of remarks available to the PC to a contextually appropriate subset that is made available to the player. Selecting a PC remark then leads to a candidate set of NPC remarks as appropriate responses to the PC. The world state is used to filter this set of remarks to a single remark, which is used by the NPC as the reply. To construct a dynamic dialogue, an author must not only create the remarks, but also write the code that determines which remarks are available to both participants at any point in the dialogue. We present "generative dialogue patterns" as a new visual language for designing dynamic dialogs and generating the program code that is necessary to select the appropriate remarks during the dialogue. We use a case study from the computer games domain to evaluate the effectiveness of generative dialogue patterns.

Keywords: dynamic dialogue, computer game, design pattern, generative, scripting.

1. Introduction

Many applications use dynamic dialogues to enable a *user* to communicate with a software system or some *agent* or aspect of a software system. In a dynamic dialog, the user and the agent alternately speak a statement, which we refer to as a *remark*. One common example is a conversation between a player character (PC) and a non-player character (NPC) in a computer game, where the PC represents the user and the NPC represents the agent. Other examples include question/answer systems in the education/training domain and web-based or application-based information/help systems. In a dynamic dialogue, there is a fundamental difference between the user and agent roles. At each turn, the user can choose between one or more potential remarks. However, when it is the agent's turn, the software system must determine a single remark (the user does not make a choice). One consequence of this asymmetry is apparent in a computer game, when two NPCs have a dialogue that is overheard by the PC. In this case, both NPCs are agents so that no user choice is necessary.

The author of a dynamic dialogue must create the set of remarks for the agent. The author must also create the set of remarks available to the user, unless a natural language processing system is employed to parse and resolve user remarks. However, even in this case, the author must map the user's natural language input to one of a finite set of choices that can be viewed as the canonical remarks available to the user.

A dynamic dialogue can be viewed as a path through a static dialogue graph. A *visual remark dialogue graph* is a static dialogue graph that contains one *agent vertex* for each agent remark and one *user vertex* for each user remark that could be made during the dynamic dialog. The graph also contains a directed arc from each vertex corresponding to a remark to all vertices corresponding to remarks that could be make immediately after that remark. We say that the first remark *precedes* the second and the second remark *follows* the first. Since the agent and user take turns, each directed arc connects an agent remark vertex to a user remark vertex or visa versa. In those cases where the author wants the agent to make a series of consecutive remarks, they can be interspersed with empty user remarks denoted by [CONTINUE].

Fig. 1 shows the visual remark dialogue graph for a conversation called "m1q1a01dock1" (*Docks*) between an NPC (agent) and the PC (user) in BioWare's Neverwinter Nights (NWN) official campaign story [1]. An oval start vertex (labeled 0) has been added. Agent vertices are drawn as pentagons and labeled with a single letter (A, B, C, …). User vertices are drawn as hexagons and are labeled by a letter and a number separated by a dash (C-1, H-1, H-2, …), where the letter is one of its preceding agent vertices. The text for the agent and user remarks is not shown in Fig. 1. However, the *textual remark dialogue graph* shown in Fig. 2 (and described later) can be used to discover the textual remarks by looking at the remarks with the same labels as those found in Fig. 1.

A remark dialogue graph contains all remarks available to the agent and the user for the dynamic dialog. At run-time, a specific path is traced through the static dialogue graph to produce a list of vertices. For example, one path through the dialogue graph shown in Fig. 1 is: 0 [START], F *You one of those damned thieves? …* , F-1 *[CONTINUE]*, G *See, I just want to go back to work …*, G-1 *[CONTINUE]*, H *Can't do squat with all the thieves …*, H-2 *Me want ask questions.*, J *I ain't gonna waste my time …*, H-5 *Goodbye.*, L *Bye then. Guess I'll just ….*

* Corresponding author. Tel.: 1 780 492 5468; fax: 1 780 492 1071.
E-mail address: duane@cs.ualberta.ca (D. Szafron)

When the dynamic dialogue is at a particular vertex in the graph, not every following vertex in the static remark dialogue graph can be selected. First, the dialogue author must provide some mechanism to enable the agent to select a single remark at any point in the dialogue. For example, at the start of the dialogue shown in Fig. 1, the agent must select exactly one of 6 potential agent remarks (A-F), as the most appropriate remark. In this example, the agent will select: A if a particular game quest has been completed and the agent is talking to another agent, B if the game quest has not been completed and the agent is talking to another agent, C if the game quest has been completed and the agent is talking to the user, and D, E, or F if the game quest has not been completed and the agent is talking to the user. The choice of D, E, or F depends on a game attribute of the PC, called charisma, where the agent selects D if the PC has high charisma, E if the PC has medium charisma and F if the PC has low charisma. The charisma is a value that is set when the user creates the PC at the start of the game and can change based on events in the game.

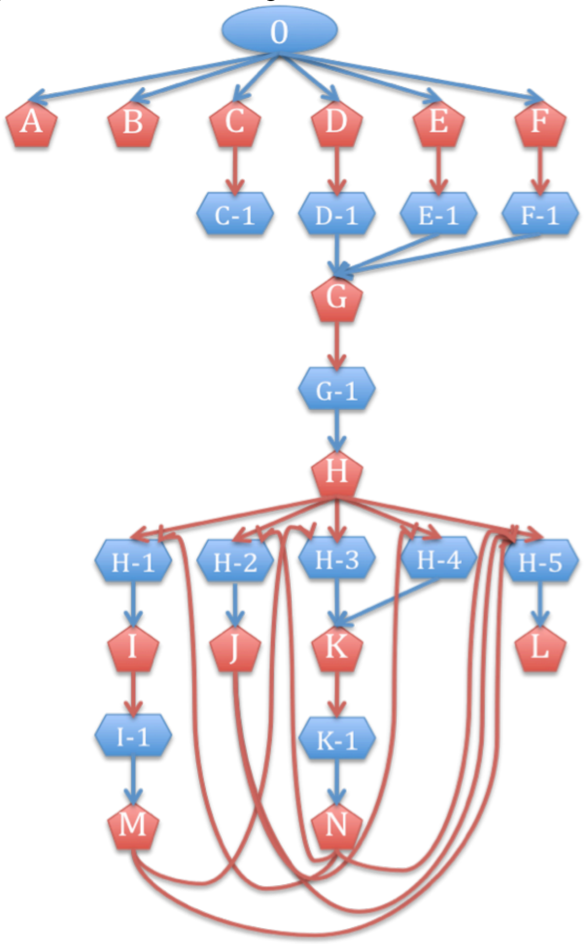
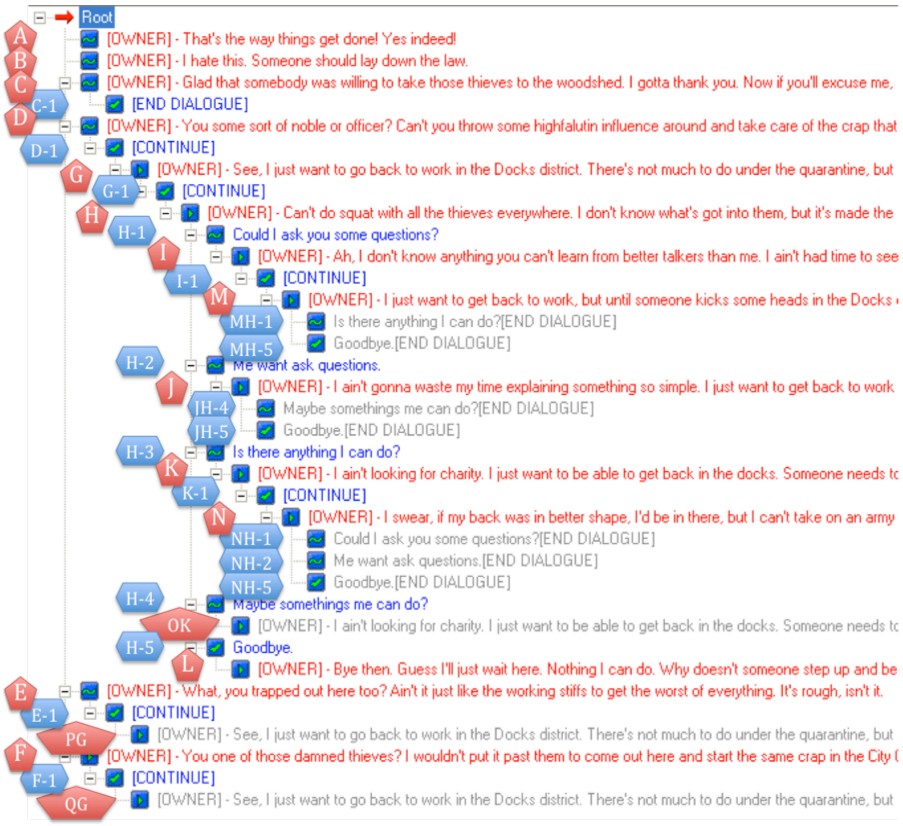


Fig. 1. A *visual remark dialogue* graph for the NWN *Docks* conversation. Petagons represent agent remarks and hexagons represent user remarks.

Second, a dialogue author must be able to limit the remarks available to the user at any time to a subset of those appearing in the static dialogue graph, depending on world state. For example, in the static graph shown in Fig. 1, vertex H *Can't do squat with all the thieves …*, has five vertices that follow: H-1 to H-5. However, only three of these vertices should appear, based on the game state. If the PC has at least normal intelligence then user vertices: H-1, H-3 and H-5 should appear and if the PC has low intelligence then user vertices: H-2, H-4 and H-5 should appear.

Therefore, in addition to creating the remarks, the author must use some *dialogue control semantics* to specify the *dialogue control*. The remark dialogue graph does not explicitly indicate the dialogue control used to filter the potential remarks based on the world state to one remark for an NPC and one or more appropriate remarks for a PC. The dialogue control must be specified separately. In most cases, dialogue control is more complex, time-consuming and error-prone than remark creation. Therefore, dialogue control should be explicitly added to the dialogue graph that the author uses to construct the dialogue.

Fig. 2 shows a *textual remark dialogue graph* that represents the same conversation as the visual remark dialogue graph shown in Fig. 1. This view is actually a screen capture of the conversation as it appears in the Aurora toolset, the utility used to author NWN game stories. However, line labels have been added to show the correspondence between lines in Fig. 2 and the vertices in Fig. 1. In the textual remark graph, an *agent line* (NPC) is labeled with [OWNER] and appears in red text in the Aurora toolset, while a *user line* (PC) does not have an [OWNER] label and appear in blue in the Aurora toolset. A line whose text is in light gray is called a *link line* and designates that the remark is elsewhere in the graph and shares the same text. The line labels in Fig. 2 correspond to the vertices with the same labels as Fig. 1. However, link lines in Fig. 2 have no corresponding vertices in Fig. 1. Therefore, each link line is an alias for a vertex in Fig. 1 and contains the line for which it is an alias as part of its label. For example, user link lines MH-5, JH-5, and NH-5 are aliases for user line H-5 and agent link lines PG and QG are aliases for agent line G. The ends of long lines are truncated so that the text is large enough to read.



```
Root
  [OWNER] - That's the way things get done! Yes indeed!
  [OWNER] - I hate this. Someone should lay down the law.
  [OWNER] - Glad that somebody was willing to take those thieves to the woodshed. I gotta thank you. Now if you'll excuse me,
    [END DIALOGUE]
  [OWNER] - You some sort of noble or officer? Can't you throw some highfalutin influence around and take care of the crap that
    [CONTINUE]
      [OWNER] - See, I just want to go back to work in the Docks district. There's not much to do under the quarantine, but
        [CONTINUE]
          [OWNER] - Can't do squat with all the thieves everywhere. I don't know what's got into them, but it's made the
            Could I ask you some questions?
              [OWNER] - Ah, I don't know anything you can't learn from better talkers than me. I ain't had time to see
                [CONTINUE]
                  [OWNER] - I just want to get back to work, but until someone kicks some heads in the Docks
                    Is there anything I can do?[END DIALOGUE]
                    Goodbye.[END DIALOGUE]
            Me want ask questions.
              [OWNER] - I ain't gonna waste my time explaining something so simple. I just want to get back to work
                Maybe somethings me can do?[END DIALOGUE]
                Goodbye.[END DIALOGUE]
            Is there anything I can do?
              [OWNER] - I ain't looking for charity. I just want to be able to get back in the docks. Someone needs tc
                [CONTINUE]
                  [OWNER] - I swear, if my back was in better shape, I'd be in there, but I can't take on an army
                    Could I ask you some questions?[END DIALOGUE]
                    Me want ask questions.[END DIALOGUE]
                    Goodbye.[END DIALOGUE]
            Maybe somethings me can do?
              [OWNER] - I ain't looking for charity. I just want to be able to get back in the docks. Someone needs tc
            Goodbye.
              [OWNER] - Bye then. Guess I'll just wait here. Nothing I can do. Why doesn't someone step up and be
  [OWNER] - What, you trapped out here too? Ain't it just like the working stiffs to get the worst of everything. It's rough, isn't it.
    [CONTINUE]
      [OWNER] - See, I just want to go back to work in the Docks district. There's not much to do under the quarantine, but
  [OWNER] - You one of those damned thieves? I wouldn't put it past them to come out here and start the same crap in the City (
    [CONTINUE]
      [OWNER] - See, I just want to go back to work in the Docks district. There's not much to do under the quarantine, but
```

Fig. 2 A textual remark dialogue graph for the NWN *Docks* conversation depicted in Fig. 1

BioWare's NWN dialogue control semantics apply to the dialogue graphs shown in Fig. 1 and Fig. 2. The control semantics uses Boolean-valued *filter* scripts to read world state and *action* scripts to set world state during dialogues and external to dialogues. A *filter* script and/or *action* script can be attached to any vertex. If a vertex has a *filter* script attached and if its Boolean expression evaluates to false, the vertex is not displayed. In addition, when more than one agent (NPC) vertex survives filtering, the lexically first unfiltered vertex is used and the others are filtered, since the agent must have a unique vertex at any time. This semantics is analogous to Java's short-circuit semantics for Boolean operators. If a script returns TRUE, all subsequent scripts are ignored. Due to the short-circuit semantics, it can be difficult to determine which of the agent sibling nodes will be displayed for a particular game state without looking at all sibling scripts.

In Fig. 2, each line that has a script is marked with the symbol ~. The *filter* scripts for vertices B and D and the action script for vertex C-1 are shown in Fig. 3. The filter script for line B checks to see that a particular quest has NOT been completed (recorded as a number in the variable, `NW_G_M1Q4MainPlot`) and whether the agent is engaged in conversation with another agent instead of the user. If both of these conditions are true then Line B is selected as the agent remark. The value of the variable `NW_G_M1Q4MainPlot` that is tested in this *filter* script could be set in the *action* script of another dialogue after the protagonist has completed the quest or set by some other script in the game. In this case, it is set when the user obtains a particular item that marks the end of the quest. After the quest has been completed, this line would never be used. The action script for line C-1 is performed after the agent makes the remark: *Glad that somebody was willing* .... This action script causes the agent to move towards an area exit with object tag `M1Q1_M1Q4A` and then to disappear as if the agent had exited the area.

Unfortunately, the *filter* and *action* script components of the dialogue control semantics do not appear explicitly in the dialogue graphs shown in Fig. 1 or Fig. 2. Dynamic dialogues in other applications may use different dialogue control semantics than the semantics used in NWN. However, whether these semantics are expressed as *filter* and *action* scripts or some other programmatic control, they are usually absent from the remark dialogue graph and the author is left to wonder about dialogue control when viewing and editing the graph.

Dialogue control semantics are difficult to write, especially for authors who are non-programmers. They are also error-prone, since they may use world state that is stored in global variables, rather than any localized scopes. Finally, they are often difficult to organize, since they can use awkward rules such as NWN's short-circuit semantics for *filter* scripts. Such rules mean that the *filter* scripts are not independent.

```
// Line B (filter script m1q1adockclg02):
int StartingConditional()
{
   int bCondition = GetLocalInt(GetModule(),"NW_G_M1Q4MainPlot")
     < 100 && !GetIsObjectValid(GetPCSpeaker());
   return bCondition;
}

// Line C-1 (action script m1q1dockslg01):
void main()
{
  ClearAllActions();
  ActionForceMoveToObject(GetNearestObjectByTag("M1Q1_M1Q4A"),
    FALSE,4.0);
  ActionDoCommand(DestroyObject(OBJECT_SELF));
  SetCommandable(FALSE);
}

// Line D (filter script nw_d2_chrh)
#include "NW_I0_PLOT"
int StartingConditional()
{
  return CheckCharismaHigh();
}
```

Fig. 3. Filter scripts that provide dialogue control for vertices B and D and an action script for vertex C-1 in the NWN *Docks* conversation shown in Fig. 1 and Fig. 2

Our goal was to create a visual language that: 1) enables a non-programmer to easily specify the dialogue control semantics, 2) displays the dialogue control semantics explicitly in the dialogue graph for easier editing and maintenance, 3) automatically generates code that implements the dialogue control semantics, and 4) organizes the remarks into higher level structures than individual remarks so that the dialogue graphs are smaller and easier to create and maintain.

To evaluate our visual programming language, we selected a specific application domain so that we could evaluate all four attributes of our goal. We picked computer game dialogues and specifically, the Neverwinter Nights game. However, this research applies to the general problem of creating dynamic dialogues. In fact it also applies to the more general problem of controlling traversals in directed graphs. One important subset of such applications is controlling game-tree search (not dialogues), in which two participants alternate moves at alternate levels of the tree. The user-agent model can be applied to game-trees in which a human player makes one set of moves and the computer makes the other and the agent-agent model works for game-trees in which the computer makes both sets of moves (self-play).

2. Related work

Our definition of dynamic dialogue limits the participants to a static set of remarks as done in NWN. The conversation is designed and built by authors before the game is played, and the player only has a few options for the PC to remark. While this is common for many computer games, it is not the only way to converse with NPCs. The Elder Scroll series [2] uses a "wiki dialogue" style of conversation. The player is presented with a statement from the NPC, and various words and phrases are hyperlinked. Clicking on these hyperlinked words brings the player to another statement about that topic. At no point does the player actually select anything for the PC to say back to the NPC and, in general, the player can explore the topics NPCs provide in any order. Since this style of dialogue does not simulate a natural conversation between two parties, it will not be discussed in this paper. Editing such dialogues is quite complex [3].

At least two game-related projects are trying to generalize dynamic dialogs beyond a dynamic path through a static dialogue graph.

Microsoft Research has investigate the application of NLP techniques to NPC dialogue [4] [5]. Each NPC has a knowledge base that changes 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 PC responses. Rather than filtering responses in a static dialogue graph, the graph itself is dynamic.

The Façade project [6] goes one step further by creating a game that provides interactive drama. Instead of giving the player only a list of pre-determined responses, the player can now use the keyboard to enter any freeform natural language statement (in English) or question to communicate with the NPCs. The NPCs have sophisticated motivational-based artificial intelligence to determine how to respond to the question and what actions to perform in the virtual world.

Other games have skirted textual dialogue entirely. For example, instead of textual sentences, The Sims [7] uses abstract symbols to show the player what information Sim characters are communicating with each other. The domain of communication is abstracted into basic wants, needs, and emotions. This abstraction allows the game to have completely dynamic conversation without complex and costly NLP techniques or the large design overhead that is associated with static content.

Other dialogue editors include SimDialog [8]. It is similar to the Aurora editor in that it constructs static dialogue graphs with alternating PC and NPC nodes. However, unlike Aurora, it allows the designer to break dialogues graphs into sub-dialogue components that represent "phases" of the conversation. These sub-dialogues can then be connected together to form the final dialogue. While it has no formal scripting language, it does have a built-in "Cause and Effect" system where conversation nodes are probabilistically selected based on one or more integer "states" ranging from -1 to 1. For example, a guard with a high (i.e. 1.0) duty state would select nodes that are weighted towards a higher duty. This allows interesting conversations to be built without the complexity of scripting, at the cost of much flexibility.

Outside of computer games, there is a vast body of research on general conversation systems and models that extend far beyond static dialogue graphs. These systems support dynamic conversation by generating sentences for the agent from a knowledge base of rules. One example is research by McRoy and Ali [9] on better computational models of dialog, specifically in the context of training and education. Another example is Traum & Larsson's Information State Approach to dialogue management [10]. General dialogue systems involve many complicated subsystems, such as a database, a planner, etc, and each dialogue system has their own approach. This makes it difficult to attempt to use a new dialogue theory with an existing dialogue system due to tight coupling of the subsystems. Traum presents the Information State Approach as a standardized form of dialogue management that allows a quicker method of implementing a new dialogue theory against an existing dialogue system. Similar to dialogue patterns, the Information State Approach dictates how a movement is performed in a dialogue, and how and when updates are made to the dialogue state. However, the similarities end there. The Information State approach is targeted towards researchers and implementers of dialogue systems who use a library such as TrindiKit to implement their dialogue theories. Its goal is to provide a standard interface and model of dialogue management as opposed to a simpler abstraction. Dialogue Patterns, on the other hand, are much simpler in scope and are intended as an abstraction for non-programmers to design a static dialogue tree with dynamic control flow. These abstract patterns also generate all the scripting code automatically, so the non-programmer does not have to understand the scripting language itself.

Our approach is inherently restricted to static dialogue graphs. However, since the vast majority of dynamic dialogs are currently based on static graphs, providing authors with better tools for creating these graphs and easily adding dialogue control semantics could have a large practical effect on dynamic dialogue creation. We have selected the BioWare Aurora conversation editor in NWN as the example of static dialogue graph editors, since it is one of the best. Other examples include conversation editors from BioWare's other games (KOTOR, Jade Empire, Mass Effect) and games built by other companies using derivatives of their engines and tools such as The Witcher by CD Projekt [11] and the Neverwinter Nights 2 Electron Toolset by Obsidian Entertainment [12]. A very different example is Conedit for the Deux Ex game from Ion Storm [13], where any conversation other than a linear one, requires extensive author work with flags used for filter control. The Planet Deux Ex website has a series of seven tutorial on using Conedit [14].

3. Script management and abstraction

For large conversations, there may be several vertices that require the same *filter* or *action* script, or a slight variation on the same script. For each vertex, 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 vertex.

It is possible that a script already exists, but the author needs to change a few parameters to accommodate the context of the vertex. For example, the Aurora toolset has a default "intelligence" script that returns TRUE if the PC's intelligence is greater than 9. The intelligence of a PC is another attribute that is set when the PC is created at the start of the game and can change as the game is played. The author may want to change the number to 11 for a specific vertex. Unfortunately, the Aurora toolset does not allow parameters to be passed to a script[1]. 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 vertices, the author must still attach the script manually for each vertex. For example, in the NWN official campaign story there is a dialogue for a

[1] A sequel game, Neverwinter Nights 2, was released on October 31, 2006.
It has an improved toolset that supports script parameters.

character named Emernick that uses 8 unique *filter* scripts (3 default, 5 custom), but those scripts are attached to 43 vertices. To use an existing script in a new vertex, the author needs to know both that the script exists and the intent of the script.

In the Aurora toolset, there is no mechanism to manage the intent or purpose of scripts other then their names. For example, the *filter* scripts in Fig. 3 are named: `m1q1adockc1g02, m1q1adocks1g01`, and `nw_d2_chrh`, which do not adequately convey their intents. In addition the filter scripts are not independent. For example, the filter script for line D of Fig. 2 seems simple. The line should be spoken if the PC (user) has high charisma. However this line has hidden conditions that are due to the short-circuit semantics of *filter* scripts. There are *filter* scripts on the sibling lines (A, B and C) that are evaluated first. If the quest has been completed, one of the filters on vertex A or C will evaluate to true. If one of the two filter scripts are removed (A or C) then the author would need to explicitly guard line D with an additional condition that the quest has not been completed. If the quest has not been completed then it is still not necessarily true that the agent should use vertex D. Vertex B has the filter shown in Fig. 3 that checks to see if the quest has not been completed and the NPC (agent) is speaking to another NPC (agent). Unfortunately, the `nw_d2_chrh` script does provide any indication that the *quest not complete* condition or the *speaking to user* conditions are necessary. Therefore, the author must consider the intent and filters of all four scripts when trying to determine the conditions necessary to display agent line D.

Since many large dialogues have 40 or more vertices with scripts attached, it becomes difficult for the author to keep track of which vertices have scripts, and the intent of these scripts. Many *action* scripts set variables on objects so that *filter* scripts can check these variables. Such script interactions require the author to mentally manage the structure of dialogues, complicating the process of fixing bugs when dialogues function incorrectly. This problem can be eliminated by explicitly including dialogue control in dialogue graphs.

4. Abstraction deficiencies in classical dialogue graphs

In Section 1 and Section 3, we described two deficiencies in classical dialogue graphs – the lack of explicit dialogue control semantics in the graph and lack of support for script management. These were the main motivations for our research. However, the representation of individual remarks as vertices also makes large dialogue graphs hard to create and understand, independent of the dialogue control issue. We will use NWN dialogue graphs to illustrate this problem. However, any dialogue graph that represents individual remarks as vertices and does not have some other abstraction or structuring mechanism for vertices will suffer from similar, although perhaps not identical problems. By focusing on a specific domain (NWN dialogues), we can objectively evaluate the relative improvements of our dialogue pattern approach. We will use NWN textual remark dialogue graphs in our discussion since this is the format that is provided to a dialogue author in the Aurora Toolset, which is used to construct these dialogue graphs.

The Neverwinter Nights official campaign story contains many large and complicated conversations, each represented by a textual remark dialogue graph similar to the one shown in Fig. 2. An example of a large conversation is the one for the Emernick character from Chapter 1 of the official campaign story. The complete graph contains 176 vertices with a depth (the maximum number of levels needed to reach any leaf vertex from the root vertex) of 10. A vertical line that connects sibling nodes denotes each level of a dialogue graph (see Fig. 2). If the graph is expanded to reveal lower levels, outer vertical lines become longer, and the sibling nodes move farther apart. Once several levels of the graph 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 dialogue. To get a better overview, the dialogue needs to be abstracted. Unnecessary details of the dialogue should be hidden so that sibling nodes of interest can move closer together.

The Aurora conversation editor only provides one mechanism to abstract the dialogue graph: expanding and collapsing the entire subgraph of a vertex. For example, in Fig. 2, the vertex represented by line D could be collapsed to move the sibling lines E and F closer to the line D. Unfortunately, collapsing a vertex hides its entire subgraph, including any important branches of dialogue that the author may want to view. For example, collapsing the subgraph under vertex D hides the structure of the hidden subgraph, including important architectural information, such as the fact that there are paths in the visible graph (lines E-1 and line F-1) that lead to line G, via indirect link lines PG and QG respectively, hidden inside the collapsed subgraph. Our goal is to hide detail, without hiding architectural (structural) information. Our approach will include collapsing long linear chains of vertices (with no branching except at the end) into a single abstract vertex. For example, we will show how vertices G, G-1, H and H-1 to H-5 from Fig. 2, can be collapsed into a single abstract vertex, without hiding any structure information.

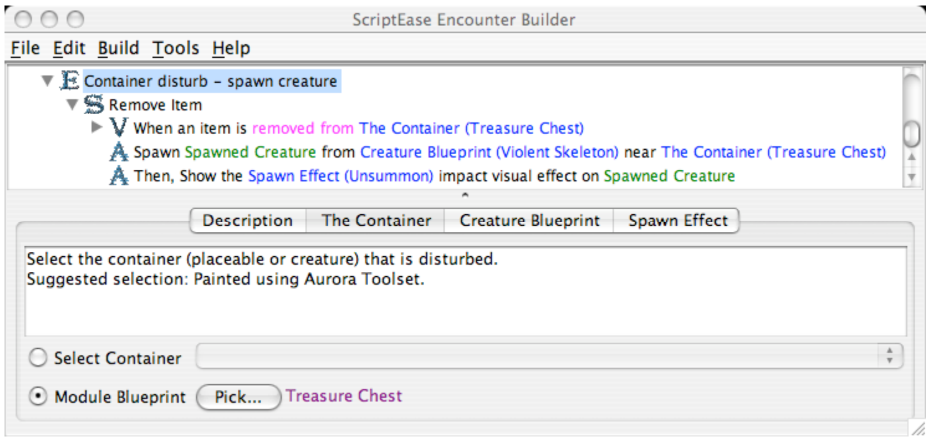
A textual remark graph contributes another problem – link lines. Recall that a link line (shown in grey in Fig. 2 and with an asterisk in its label) represents an indirection (link) to a line (vertex) that appears elsewhere in the graph. For example, the *Docks* conversation shown in Fig. 2 has 10/36 lines (28%) that are link lines. The larger "Emernick" conversation has 51% link lines. When inspecting a link line, it is very difficult for an author to determine the location of the real line. With the Aurora toolset, if the user double clicks on the link line, the focus of the window is shifted to the real line, which is called the target. Unfortunately, if the graph is large, the original link line will disappear from view in the window. The lack of context between a link line and its target is disconcerting and confusing. In a graphical view this would correspond to the need to scroll the window to follow an arc from a source vertex to a target vertex. The real problem is a lack of abstraction that can remove unnecessary detail so that the source and target of an arc are both visible in the window at the same time. Any layout program for a dialogue graph may be able to keep the source and target vertices of any one arc close enough so that no scrolling is necessary to see both vertices. However, in general it is impossible to place all source and target vertices of all arcs close together.

We use patterns [15] as our abstraction mechanism for dialogue graphs. Since we want to support dynamic dialogue control, we use generative patterns so that the dialogue control can be generated automatically from the patterns.

5. ScriptEase generative design patterns

ScriptEase is a tool that uses generative design patterns to generate NWScript code for the Neverwinter Nights (NWN) computer role-playing game [16]. Instead of manually writing scripts, the author instantiates a pattern and adapts it to specific game scenario. ScriptEase can then automatically generate the scripting code from the pattern. Currently, ScriptEase supports *encounter patterns* for player-object interactions [17] and *behaviour patterns* for non-player character (NPC) behaviours [18]. For example, *container disturb spawn creature* is an example of an encounter pattern that has three options (parameters): a container, the kind of creature to spawn and a visual effect that should occur. If the author creates an instance of this pattern, and binds its options to specific objects in a story: *treasure chest*, *violent skeleton*, and *unsummon*, then when the player character (PC) tries to remove any item from the treasure chest, a violent skeleton will be spawned and the unsummon visual effect will occur. Fig. 4 shows such a pattern instance in ScriptEase, opened to reveal its components. Fig. 5 shows part of the NWScript code generated by the pattern and Fig. 6 shows a screenshot from the story just as the PC removes an item from the chest to trigger the encounter pattern instance.

An example behavior pattern is a *guard*, who guards an item stored in a container. This behavior pattern generates NWScript code that grants the guard three proactive independent behaviors: *patrol* near the chest, *rest* on a nearby bench, *check* that the guarded item is still in the chest and one proactive collaborative behavior, *converse-talk* with a friendly NPC. If there are not external stimuli, the guard selects one of these proactive behaviors probabilistically, biased by current motivation levels: *duty*, *tiredness* and *threat*. The guard also has one reactive collaborative behavior that can be initiated by another friendly NPC called *converse-listen*. Finally, the guard also has two latent behaviors that can be triggered by the PC or an unfriendly NPC, *warn* an intruder that gets too close to the chest and *attack* an intruder that opens the chest. The visual representation and NWScript code generated by a behavior pattern is similar to the representation and script generated by an encounter pattern and shown in Fig. 4 and Fig. 5 respectively.

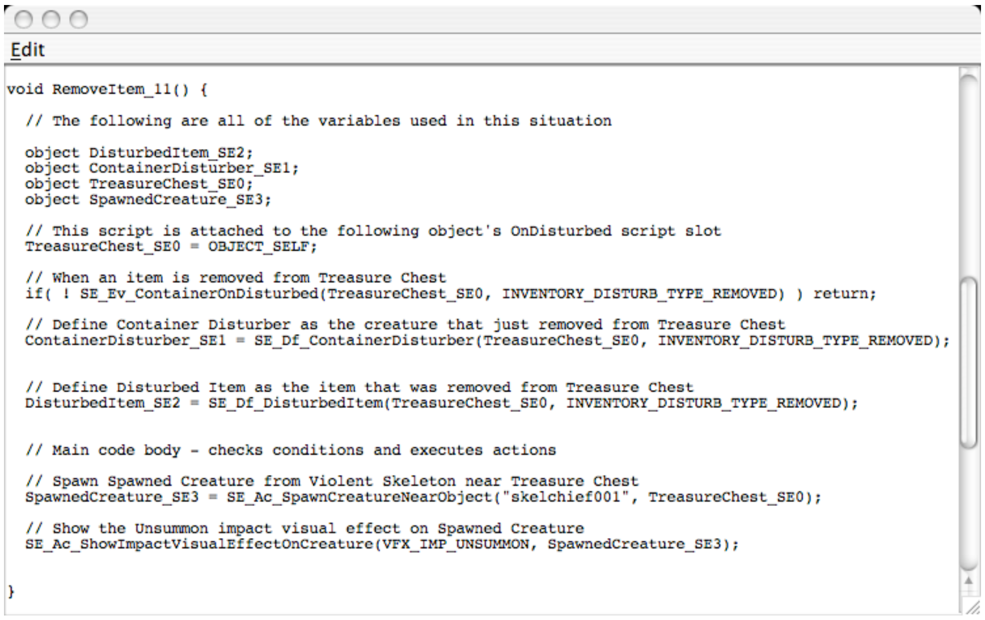


E Container disturb – spawn creature
S Remove Item
V When an item is removed from The Container (Treasure Chest)
A Spawn Spawned Creature from Creature Blueprint (Violent Skeleton) near The Container (Treasure Chest)
A Then, Show the Spawn Effect (Unsummon) impact visual effect on Spawned Creature

Select the container (placeable or creature) that is disturbed.
Suggested selection: Painted using Aurora Toolset.

( ) Select Container
(•) Module Blueprint Treasure Chest

Fig. 4. A ScriptEase Encounter pattern, opened to reveal its components.



```
void RemoveItem_11() {

  // The following are all of the variables used in this situation

  object DisturbedItem_SE2;
  object ContainerDisturber_SE1;
  object TreasureChest_SE0;
  object SpawnedCreature_SE3;

  // This script is attached to the following object's OnDisturbed script slot
  TreasureChest_SE0 = OBJECT_SELF;

  // When an item is removed from Treasure Chest
  if( ! SE_Ev_ContainerOnDisturbed(TreasureChest_SE0, INVENTORY_DISTURB_TYPE_REMOVED) ) return;

  // Define Container Disturber as the creature that just removed from Treasure Chest
  ContainerDisturber_SE1 = SE_Df_ContainerDisturber(TreasureChest_SE0, INVENTORY_DISTURB_TYPE_REMOVED);


  // Define Disturbed Item as the item that was removed from Treasure Chest
  DisturbedItem_SE2 = SE_Df_DisturbedItem(TreasureChest_SE0, INVENTORY_DISTURB_TYPE_REMOVED);


  // Main code body - checks conditions and executes actions

  // Spawn Spawned Creature from Violent Skeleton near Treasure Chest
  SpawnedCreature_SE3 = SE_Ac_SpawnCreatureNearObject("skelchief001", TreasureChest_SE0);

  // Show the Unsummon impact visual effect on Spawned Creature
  SE_Ac_ShowImpactVisualEffectOnCreature(VFX_IMP_UNSUMMON, SpawnedCreature_SE3);


}
```

Fig. 5. Some of the NWScript code generated by the ScriptEase encounter pattern in Fig. 4.

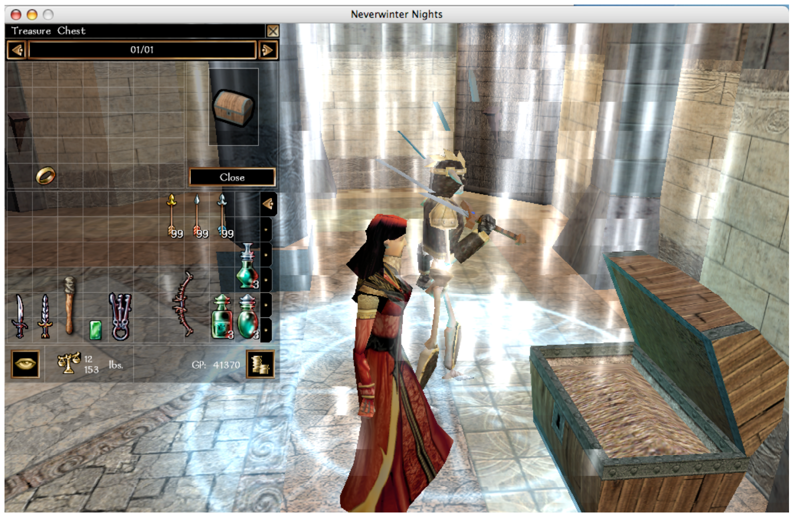


Fig. 6. The game story effect of a disturb container spawn creature pattern

In this paper we introduce a new kind of generative design pattern – the *dialogue pattern*. Although we use computer games to illustrate this new pattern type, dialogue patterns can be used for dynamic dialogues in any application. Dialogue patterns are different from the encounter and behaviour patterns currently available in ScriptEase. Dialogue patterns are more structure-based than intent-based. Nevertheless, our dialogue patterns can be integrated with the ScriptEase tool and its existing patterns for use in computer games. We are currently implementing dialogue patterns into a functional dialogue editor in ScriptEase.

6. Dialogue graphs

A remark graph contains only three kinds of components, agent remark vertices, user remark vertices and connection arcs. A *dialogue graph* is composed of eighteen different kinds of components and sub-components. At the highest level, a dialogue graph is composed of *dialogue patterns* and each dialogue pattern consists of *topics* and *decisions*, where a topic encapsulates a coherent sub-graph of the dialogue dealing with a single issue and a decision is used by an agent to decide which topic should be presented at a particular point in the dialogue. However, the definition of a dialogue pattern is recursive, since dialogue patterns can ultimately contain other dialogue patterns. A topic contains a hierarchy of *structural components* that abstract the vertices and arcs of remark dialogue graphs: *exchanges, agent remarks, user menus, user choices*, *user remarks, user text, user links, user choice groups*, *agent menus, shared user choices, shared user menus, user link icons*, and *topic icons*. In addition, dialogue patterns contain *control components*: *user choice filters*, *agent remark* *decisions*, *decision icons, global user remark filters*, and *decision outcomes* that together with decisions, use *filters* to generate filter scripts that control the dynamic appearance of dialogue components at run-time. These control components have no visual analogue in remark dialogue graphs. An Extended Bachus-Naur form grammar for a dialogue graph is shown in Fig. 7.

```
Terminals: ROOT, SUMMARY, TEXT, NAME, NIL, TRUE, FALSE
dialogue-graph ::= root dialogue-pattern
root ::= ROOT [global-filter-pattern]
global-filter-pattern ::= NAME global-clause { global-clause }
global-clause ::= conditions TEXT
dialogue-pattern ::= topic | decision | NIL
topic ::= SUMMARY { exchange } exchange
exchange ::= agent-remarks user-menu
user-menu ::= { user-filter-pattern } { choice }
user-filter-pattern ::= NAME user-filter-clause { user-filter-clause }
user-filter-clause ::= conditions { user-filter-pattern } { remark }
choice ::= remark dialogue-pattern
remark ::= TEXT | global-filter-pattern
agent-remarks ::= TEXT | agent-filter-pattern
agent-filter-pattern ::= NAME agent-filter-clause { agent-filter-clause } agent-otherwise-clause
agent-filter-clause ::= conditions { agent-filter-pattern } TEXT
agent-otherwise-clause ::= { agent-filter-pattern } TEXT
decision ::= SUMMARY decision-filter-pattern
decision-filter-pattern ::= NAME decision-filter-clause { decision-filter-clause } decision-
     otherwise-clause
decision-filter-clause ::= conditions { decision-filter-pattern } decision-outcome
decision-otherwise-clause ::= { decision-filter-pattern } decision-outcome
decision-outcome ::= dialogue-pattern
conditions ::= boolean { boolean }
boolean ::= TRUE | FALSE
```

Fig. 7 An EBNF grammar for a dialogue pattern.

The NWN conversation that is represented by the dialogue graphs shown in Fig. 1 and Fig. 2 can be represented by the dialogue pattern shown in Fig. 8. User and agent remark labels from Fig. 2 have been added to Fig. 8 to illustrate the correspondence and the layout has been changed to fit on one page for this paper. The layout normally has no crossing lines, but takes up more screen real estate. Section 7 describes the structural components shown in Fig. 8 and Section 8 describes the control components.

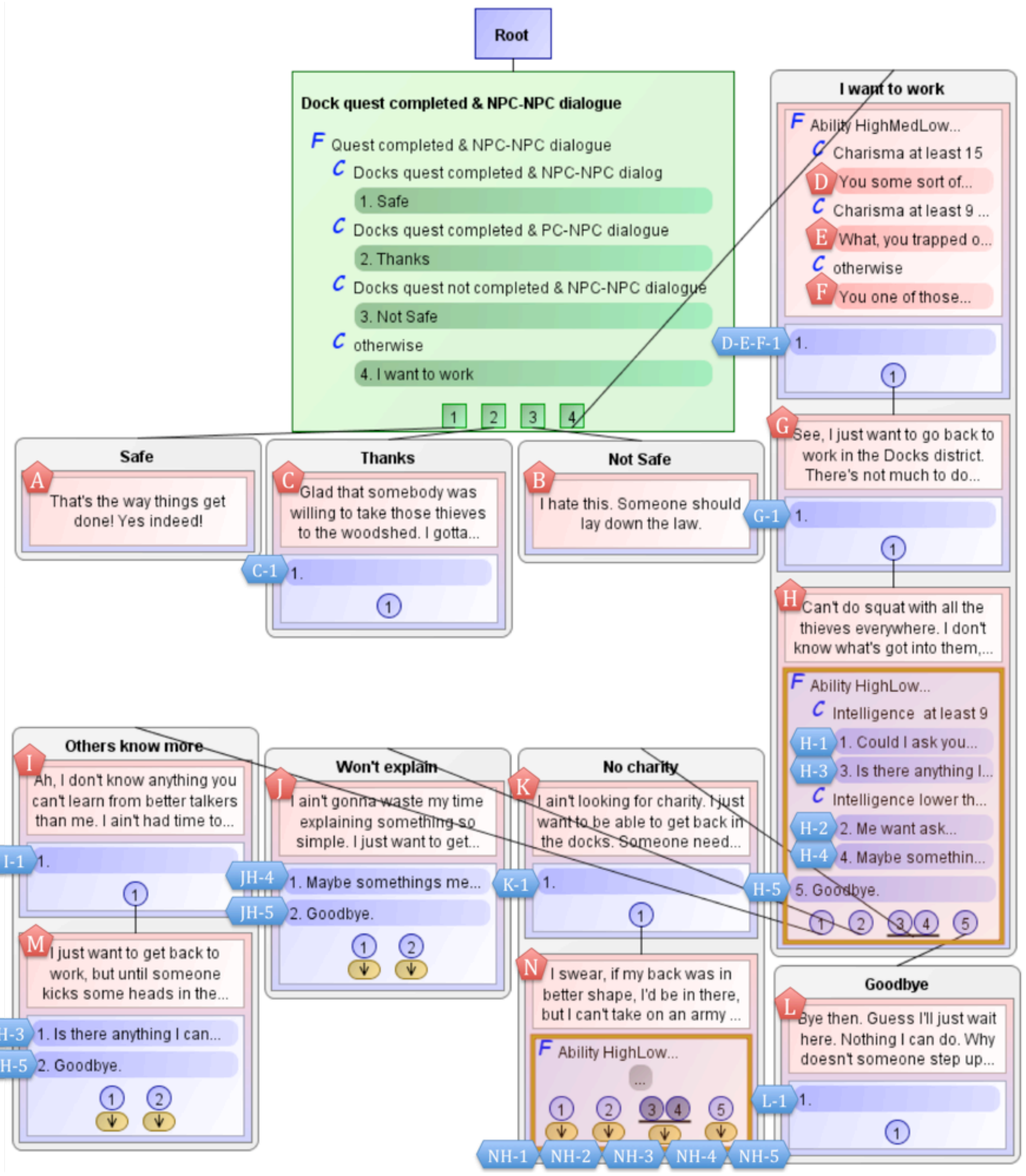

Root
Dock quest completed & NPC-NPC dialogue
F Quest completed & NPC-NPC dialogue
C Docks quest completed & NPC-NPC dialog
1. Safe
C Docks quest completed & PC-NPC dialogue
2. Thanks
C Docks quest not completed & NPC-NPC dialogue
3. Not Safe
C otherwise
4. I want to work
I want to work
F Ability HighMedLow...
C Charisma at least 15
D You some sort of...
C Charisma at least 9 ...
E What, you trapped o...
C otherwise
F You one of those...
D-E-F-1 1.
G See, I just want to go back to work in the Docks district. There's not much to do...
G-1 1.
H Can't do squat with all the thieves everywhere. I don't know what's got into them,...
F Ability HighLow...
C Intelligence at least 9
H-1 1. Could I ask you...
H-3 3. Is there anything I...
C Intelligence lower th...
H-2 2. Me want ask...
H-4 4. Maybe somethin...
H-5 5. Goodbye.
Safe
A That's the way things get done! Yes indeed!
Thanks
C Glad that somebody was willing to take those thieves to the woodshed. I gotta...
C-1 1.
Not Safe
B I hate this. Someone should lay down the law.
Others know more
I Ah, I don't know anything you can't learn from better talkers than me. I ain't had time to...
I-1 1.
M I just want to get back to work, but until someone kicks some heads in the...
MH-3 1. Is there anything I can...
MH-5 2. Goodbye.
Won't explain
J I ain't gonna waste my time explaining something so simple. I just want to get...
JH-4 1. Maybe somethings me...
JH-5 2. Goodbye.
No charity
K I ain't looking for charity. I just want to be able to get back in the docks. Someone need...
K-1 1.
N I swear, if my back was in better shape, I'd be in there, but I can't take on an army ...
F Ability HighLow...
NH-1 NH-2 NH-3 NH-4 NH-5
Goodbye
L Bye then. Guess I'll just wait here. Nothing I can do. Why doesn't someone step up...
L-1 1.

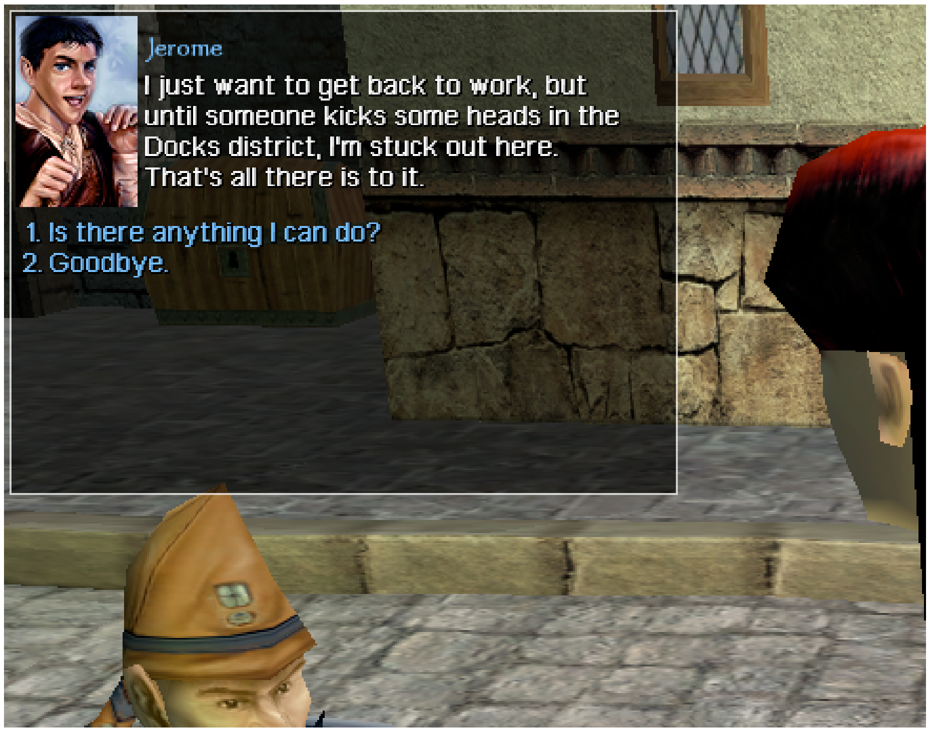
Fig. 8 A pattern dialogue graph for the NWN *Docks* conversation

7. Dialogue pattern structural components

An author uses structural components to construct and represent the static structure of a dialogue that includes all potential dialogue paths. An author requires a palette of structural components to view the dialogue at varying levels of abstraction – hiding details for prospective and context and revealing details for editing.

7.1 Exchanges – Agent Remarks, User Menus, User Choices, User Remarks and User Links

We introduce an exchange that tightly couples an agent conversation vertex with the user vertices that follow it. For example, in NWN, each conversation is presented as a series of dialogue boxes that contains one agent (NPC) remark and a user menu that contains the appropriate user (PC) remarks as choices. Fig. 9 shows the NWN dialogue that contains agent remark M and user remarks MH-3 and MH-5 in the *Docks* conversation. Question/answer and information/help systems also use similar groupings consisting or one agent remark and a menu of user remarks. Therefore a dialogue author usually thinks at the level of exchanges rather than the individual remark level and any tool or language that supports dialogue authoring should support this abstraction directly.


Jerome

I just want to get back to work, but until someone kicks some heads in the Docks district, I'm stuck out here. That's all there is to it.

1. Is there anything I can do?
2. Goodbye.

Fig. 9. The dialogue for the Exchange containing agent remark M and user remarks MH-3 and MH-5 from the remark graph of Fig. 1 and Fig. 2 or the pattern graph of Fig. 8 from the NWN *Docks* conversation

An *exchange* is represented by a grey rectangle (Fig. 8) and contains one *agent menu* and one *user menu*. An *agent menu* is represented by a red rectangle inside an exchange (Fig. 8) and contains all of the potential *agent remarks*, even though only one of them will actually appear at run-time. We refer to an exchange by the agent remarks that it contains, such as A, G or D-E-F in Fig. 8.

An *agent remark* contains agent text and depending on the application it could also contain other information such as animation and sound for the agent that makes the remark. An agent remark has the same properties as an agent vertex in a traditional vertex dialogue graph (Fig. 1). The text for the agent remark appears inside the red rectangle representing the agent menu. For example, the agent remark labeled M in Fig. 8 has text *I just want to get back to work, ….*

A *user menu* is represented by a blue rectangle inside an exchange (Fig. 8) and contains a list of *user choices*. For example, exchange M in Fig. 8 has a user menu that contains two user choices, MH-3 and MH-5.

A *user choice* represents one choice that a user can make in an exchange. It consists of a *user remark* that represents what the user says and a *user link* that indicates how the dialogue proceeds.

A *user remark* contains a number and the *user text* that is displayed. Similarly to an agent remark, a user remark can also include animation and sound for an avatar representing the user and other information germane to the remark. The text for each user remark appears in a blue rounded rectangle. For example, exchange M in Fig. 8, contains two user remarks numbered 1 and 2 that determine the lexical order for these user remarks as they appear to the user, as shown in Fig. 9. These user remarks are also annotated with MH-3 and MH-5 to show the correspondence to the labels used in Fig. 2.

If the author leaves a user remark blank (no text) then a dialogue tool or a dialogue engine can fill in a default remark. For example, the Aurora Toolset displays the text [CONTINUE] if a user vertex has no text and if that vertex does not end the conversation and displays the text [END DIALOGUE] if the vertex is empty and ends the conversation and the Aurora game engine inserts these markers into the dialogue during the game. In the pattern graph, we display blank text as blank and let the underlying application interpret blank user remarks (for NWN the appropriate remark text is generated at run-time by the Aurora game engine).

A *user link*, represented by a numbered circle and an arrow (Fig. 8), indicates which dialogue component should be used next if the user selects the user choice that contains it. A user link can target an exchange, a topic (Section 7.2) or a decision (Section 8.1). The number in the circle corresponds to the number of its corresponding user choice. For example, in Fig. 8 the user link for user choice 1 in exchange I, targets exchange M.

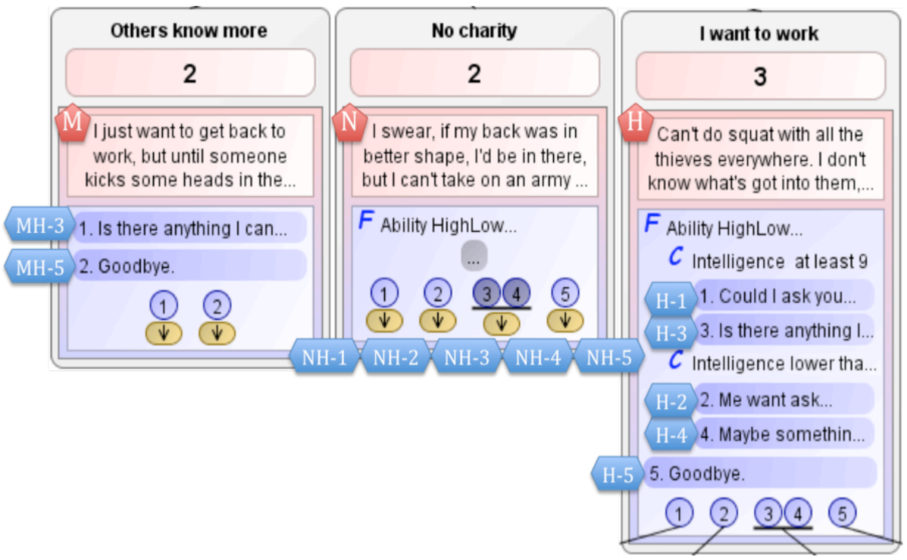
The pattern graph of Fig. 8 highlights a major advantage of exchanges. Exchange H contains user choices 1-5. Each user choice has a user link to the next dialogue component. Regardless of the size of the components, even if the components were composed of tens or hundreds of exchanges, the five user remarks for these choices would still remain spatially close together inside the first exchange. They will not become separated if the dialogue sub-graphs below this exchange grow in size such as they are in the textual remark graph shown in Fig. 2, where the remarks, H-1 to H-5 are separated by many sub-tree vertices.

7.2 Topics

An exchange contains multiple remarks and the intent of the exchange is not always clear from the text. In addition, sometimes it is necessary to read the text of other nearby exchanges to determine this intent. We introduce the notion of a topic so that the intent of portions of a dialogue graph can be abstracted and made explicit through a topic label and through the ability to group certain exchanges together into a single topic. It would be possible to provide every exchange with an "intent label" instead of introducing a new topic component. However, a topic is a more powerful abstraction than using an "exchange intent", since a topic can be used to group related exchanges together and provide a "group intent". In complex dialogues, the agent may have a lengthy explanation for the user. Instead of putting all the text into a single agent vertex, it is common practice to split the explanation across multiple agent vertices. This reduces the amount of text the user has to read at any one point in the dialogue. For example, the two exchanges I and M in Fig. 8 are two parts of a single agent explanation included in a single topic called *Others know more*. Similarly the three exchanges D-E-F, G and H are enclosed in a single topic (*I want to work*) and the two exchanges K and N comprise the topic (*No charity*). Each topic in a pattern graph is represented by a gray rounded rectangle that contains a label as well as its list of exchanges. A *topic* is defined as:

1. a labeled list of one or more exchanges where,
2. the last exchange in a topic is called the *tail exchange* and it may have one or more user choices, and
3. all exchanges except for the tail exchange are called *inner exchanges* and they must have exactly one user choice (or one user choice group – Section 7.3) that links to the next exchange in that topic.

Based on this definition, each exchange in a pattern graph can be considered as part of a topic. With each exchange now encapsulated in a topic, the pattern graph can be further abstracted without losing any structure information. Topics can be collapsed to hide all inner exchanges in the topic and leave the tail exchange visible. In a collapsed state, a number, called the *exchange count*, is placed above the tail exchange to indicate the total number of exchanges in the topic. For example, Fig. 10 shows the collapsed version of the *I want to work* topic, which contains 3 exchanges and the collapsed versions of the *Others know more* and *No charity* topics which contain two exchanges each.



Others know more
2
M
I just want to get back to work, but until someone kicks some heads in the...
MH-3
1. Is there anything I can...
MH-5
2. Goodbye.
1
2
No charity
2
N
I swear, if my back was in better shape, I'd be in there, but I can't take on an army ...
F Ability HighLow...
...
1
2
3
4
5
NH-1
NH-2
NH-3
NH-4
NH-5
I want to work
3
H
Can't do squat with all the thieves everywhere. I don't know what's got into them,...
F Ability HighLow...
C Intelligence at least 9
H-1
1. Could I ask you...
H-3
3. Is there anything I...
C Intelligence lower tha...
H-2
2. Me want ask...
H-4
4. Maybe somethin...
H-5
5. Goodbye.
1
2
3
4
5

Fig. 10. Three collapsed topics from the *Docks* conversation.

When a topic is collapsed, the number of vertices in the pattern graph is reduced without losing any information about the structure (branching) of the graph or the number of exchanges in the graph. When a topic is collapsed, the author can no longer see the remark text of the hidden exchanges. However, a visible topic label that depicts the topic intent should obviate this problem since the author can use it to quickly summarize the intent of the remarks in the topic. The author can also expand and collapse a topic as more or less detailed information is needed about the hidden remarks. In many cases this topic intent label can be more useful than scanning the remarks. Remark based dialogue graphs/tools, such as the Aurora conversation editor, do not support an intent label, so the author must scan the full remark text for many agent vertices. These remarks are often extremely verbose, with the intent found either in the middle or at the end of the series of agent remarks. Topic intents allow the author to quickly scan the conversation to find particular topics. For example, in Fig. 8, all three exchanges in the collapsed *I want to work* topic relate to the NPC indicating that he wants to work, but cannot, due to a problem in the docks. Without a topic intent, to discover (or re-discover) this intent, an author would have to read all of the remarks.

If a user choice links to an exchange, it must be the next exchange in the same topic. If an author wants to link a user choice to an exchange that is not in the same topic, the user choice must instead link to a topic that contains that exchange as its first exchange. If an author wants to link to an exchange that is not the first exchange in a topic, the topic must be split to form a new topic with the exchange as its first exchange. In this case, splitting a topic is desirable since if there is an entry into the interior of a topic, then the part of the topic after the entry should be capable of being understood independently of the part before the entry. In other words, the parts should naturally be viewed as two independent topics. A topic is like a basic block in compiler optimization – it has one entry point (at the start).

7.3 User Choice Groups

Exchanges in which different user remarks lead directly to the same agent remark are common. For example, in the dialogue graphs shown in Fig. 1, Fig. 2 and Fig. 8, user choices H-3 and H-4 lead to the same agent remark, K. In other words, it doesn't matter whether the user selects H-3. *Is there anything I can do?* or H-4 *Maybe somethings me can do?*, the next agent remark should be K, *I ain't looking for charity….* In computer games, this technique of providing multiple user choices that link to the same agent remark is often used to increase the user's sense freedom in the story without increasing the complexity of the dialogues. For example, in Chapter 1 of the NWN campaign story, this technique is used 499 times.

In Fig. 1, sharing is accomplished using two arcs, one between vertices H-3 and K and one between vertices H-4 and K. In Fig. 2, sharing is more difficult. Line H-3 is connected to line K (by preceding it lexically), but a link line is necessary to connect line H-4 to line K through link line OK. Both of these approaches can be improved. In the interests of simplifying the graph, it would be useful to combine all of these arcs with the same target into a single arc. A *user choice group* is a group of user choices in a single exchange that link to the same dialogue component. In Fig. 8, a dark line under a set of contiguous user links is used to demark a user choice group and a single link from this line to the next dialogue component replaces all of the links from individual user links in the group to the same next dialogue component. For example, user choices 3 and 4 in exchange H form a user choice group with a single arc leading to the *No charity* topic whose initial exchange is K. This approach has the advantage that if an author changes this single link to another dialogue component, it is changed for all of the user choices in the group. This is easier than changing each individual link, which would be required in the visual remark graph of Fig. 1 and much easier than changing the targets in the textual remark graph of Fig. 2, where both a direct target (line) and a link line must be changed.

7.4 Shared User Choices, Shared User Menus and Link Icons

In Section 7.1, we discussed the advantages of using explicit exchanges in dialogue graphs. However, the pattern graph of Fig. 8 also highlights a major disadvantage of exchanges. Without other abstraction techniques, exchanges cause duplication of user choices. For example, in Fig. 8, there are four exchanges that contain the user choice *Goodbye*, labeled H-5, MH-5, JH-5 and NH-5. In each case the user link of this user choice should be the topic *Goodbye*. If user choices have the same user remark and the same user link, we provide a mechanism called *shared user choices* so that this information can be shared. Notice that there is a direct link from H-5 to the topic *Goodbye*. In fact, H-5, MH-5 JH-5 and NH-5 are *shared user choices*. If the cursor is placed over any of these four user choices all four are highlighted and the shared link to the *Goodbye* topic becomes apparent. One of the shared user choices is always in *focus* so that its link is visually connected to another component. In this case, H-5 is the focus user choice and its link is connected to the *Goodbye* topic. All other copies of a shared user choice are *unfocused* and have their links replaced by a *link icon*, denoted by an arrowhead. If the link target is a topic, the arrowhead is surrounded by a yellow rounded rectangle. If the link target is a decision (see Section 8.1), the arrowhead is surrounded by a yellow rectangle. All of the link icons in Fig. 8 target a topic.

An unfocused user choice can be *focused* by right-clicking on the arrowhead icon and selecting focus from the pop-up menu. In this case, the previously focused user choice becomes unfocused and its link is replaced by the arrowhead icon. For example, if the unfocused user choice MH-5 is focused, then the *Goodbye* topic is moved below the *Others know more topic* and the link icon of HM-5 is replaced by a link from user choice MH-5 to this topic. Also, the link for H-5 is replaced by a link icon. If the user remark text or the user link target of any copy of a shared user choice is changed, all the others shared user choices are automatically updated. The other sets of shared user choices in Fig. 8 are {H-3, MH-3}, {H-4, JH-4}, {H-1, NH-1} and {H-2, NH-2}. When one user choice is shared between exchanges, it is common to share other user choices as well. Therefore, we support *shared user menus*, where two exchanges can share multiple user choices by sharing an entire user menu. Since it is common to want to share multiple user choices, but not all user choices, we allow an author to *disable* the shared user choices that are not applicable to any particular exchange. For example, in Fig. 8, exchanges H and N share a user menu. However, since user choices HN-3 and HN-4 should not appear in exchange N, when the exchange is displayed in the game, they have been disabled (dimmed). If any component of a shared user menu is changed, the component is automatically updated in all shared copies. For example, if another user choice, H-6 was added to user menu H, a shared user choice, would be added to user menu N. Only one copy of a shared user menu is in *focus*, displaying its component filters and user remarks, such as the user menu in exchange H. All other shared copies are *unfocused* with components replaced by ellipsis, such as the user menu in exchange N. However, if the user menu is clicked then all copies are highlighted (exchanges H and N in Fig. 8) so the shared components can be viewed in the focused user menu,. A user menu can be *focused* by right-clicking on the user menu and selecting focus from the pop-up menu. In this case, the previously focused user menu becomes unfocused and its components are replaced by ellipsis.

By supporting sharing at two different levels of abstraction, user menu and user choice, the author not only has the ability to choose the abstraction level based on the amount of sharing between exchanges, the author also gains more flexibility with shared dialogue control, as explained in Section 8.

7.5 Topic Icons

Pattern dialogue graphs have two mechanisms to support dialogue sub-graph re-use by allowing multiple user choices to link to the same topic – shared user choices and shared user menus. These mechanisms work well for sharing user choices. However, occasionally an author wants two user choices that have different user remarks but the same user link target. This situation does not occur for the pattern dialogue graph of the NWN *Docks* conversation described in this section. However, it does occur in other NWN conversations used in the case study of Section 9. As a concrete example, suppose that the author would like to change the user remark text in the user choice JH-5 to *Me go now*, since this is supposed to be the farewell text for a PC with low intelligence. In this case, since JH-5 is a shared user choice, the author would have to unshare this user choice before editing the user remark to prevent the text from changing in the partner user choices, H-5, MH-5, and NH-5. Suppose however that the author would like the user link of JH-5 to remain connected to the *Goodbye* topic. One possibility is to draw a direct user link from user choice JH-5 to topic *Goodbye* as soon as JH-5 is unshared. While it is always possible to draw multiple direct links from different user choices to the same topic, it is not always possible to do this in clear way that does not confuse the author. A direct link from a user choice to a topic is analogous to an arc from a user vertex (rounded rectangle) to an agent vertex (rectangle) that appears in the visual remark dialogue graph of Fig. 1 and analogous to an agent line (labeled by integer dash integer) that is directly followed by a user line (labeled by an integer) in the textual remark dialogue graph shown in Fig. 2. In a textual remark dialogue graph only one direct link is possible between a user vertex and an agent vertex, since a direct link is represented lexically by indenting the following line of text. For example, in Fig. 2, there is a direct link from user line D-1 to agent line G. Other links to the same vertex must use some form of indirect link. For example, in Fig. 2, there are indirect links from line E-1 and line F-1 to line G that pass through the link lines PG and QG. Recall from Section 4, that link vertices can cause context switching (lost focus) problems for the author.

Direct links are clearer and easier to understand than indirect links, since for an indirect link, the author must infer the location of the target. For example, in Fig. 1 there is a direct link from vertex E-1 to vertex G. The PG link vertex that appears between vertices E-1 and G in Fig. 2 does not exist in Fig. 1. Nevertheless, direct links can also cause problems. They are only useful when the user remarks that link to the same target are all spatially close to the target topic in the rendered graph. Otherwise, direct links can be problematic, since the arcs can intersect with other components. It is not as easy to follow the link from vertex H-4 to vertex K in Fig. 1, as it is to follow the link from vertex H-3 to vertex K. When a link crosses another link or vertex, the author can be confused. A *planar graph* is a graph that can be drawn so that no arcs intersect and a *nonplanar graph* cannot be drawn without arc intersections [19]. Since not all graphs are planar, in general, there is no way to guarantee that an arc does not cross another arc in a dialogue graph, so some form of indirection is necessary.

For example, can the visual remark dialogue graph shown in Fig. 1 be drawn as a planar graph? We used an implementation [20] of the *Leda* [21] algorithms to test the planarity of the graph in Fig. 1 and found that it is not planar, since there is a K3,3 sub-graph whose nodes are (H, J, M) and (H-1, H-3, H-5). When a non-planar visual remark graph is transformed to a pattern graph, it can become planar due to sharing. This is what happened to the *Docks* conversation graph when it was transformed from Fig. 1 to Fig. 8. However, we wanted to discover in practice, how many visual remark graphs can be drawn as planar graphs (only direct links necessary) and how many would require indirect links before transformation to pattern graphs? The answer to this question may be highly application dependent. However, to get some idea in the context of computer game dialogues, we converted all of the textual remark dialogue graphs for Chapter 1 of the BioWare NWN official campaign story into visual remark dialogue graphs and used Leda to test their planarity. We found that 128 of the 265 visual remark dialogue graphs (48.3%) are non-planar. This means that indirect links would be required for a significant proportion of the dialogues in any authoring tool that does not allow crossing links and indicates that our dialogue patterns must support either indirect links or other sharing mechanisms if we wish to maintain non-crossing links.

We introduce a general indirection mechanism between user choices and topics into our pattern graphs, to cover indirections that are not eliminated by shared user choices or shared menus. A *topic icon* is a yellow rounded rectangle labeled by a topic name. A user choice that links to a topic icon is an indirect link to the topic with the same name. If we return to the example, where the author changed the user remark text in the user choice JH-5 to *Me go now*, Fig. 11shows what the topic icon for the *Goodbye* topic would look like. Just as with shared user choices, the author can change focus to

JH-5 using a menu. In this case, JH-5 would be directly connected to a *Goodbye* topic that would appear directly under it and H-5 would be linked to a *Goodbye* topic icon that would appear under it. In this example, a layout algorithm may actually use direct links from both user choice H-5 and user choice JH-5 without intersecting arcs in the graph. However, we have included a topic icon to illustrate its use.

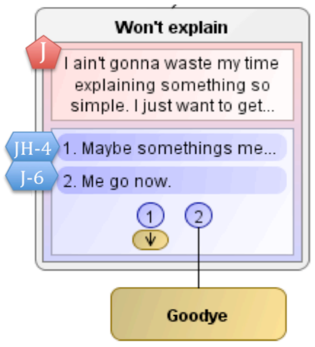

Won't explain
J
I ain't gonna waste my time explaining something so simple. I just want to get...
JH-4
1. Maybe somethings me...
J-6
2. Me go now.
1
2
Goodye

Fig. 11 A topic icon.

8. Dialogue pattern control components

An author uses control components to construct and represent the dynamic control of a dialogue. This dynamic control specifies which of all potential paths through the static dialogue graph will be exercised under the conditions that prevail when the dialogue is being performed. It is important that an author can explicitly visualize the control as an integral part of the dialogue graph and change the control by manipulating the graph.

We introduce filter patterns as a mechanism to explicitly represent dynamic control in our pattern graphs. Each *filter pattern* contains a set of options, a list of clauses, where each *clause* contains a condition and a list of zero or more *outcomes*. For example, consider the *Ability HighLow* filter that has two options: an enumeration of the abilities of the PC (*Strength*, *Dexterity*, *Constitution*, *Intelligence*, *Wisdom* or *Charisma*), and a threshold number. The pattern contains two clauses. The first clause tests wither the ability of the PC specified by the first option is strictly greater than the threshold. The second clause tests whether the ability is less than or equal to the threshold. *Ability HighLow* is a popular filter. In Chapter 1 of the NWN official campaign story there are 5,226 pairs of user vertices where this single pattern could be used to generate the scripts with ability option set to *Intelligence* and threshold set to 9 (the default values for the pattern).

If the condition of a clause evaluates to true at run-time then the outcomes in that clause will be used. A filter pattern clause is similar to a case statement clause or an if-statement clause of a general-purpose programming language. There are two fundamental differences. First, a general-purpose programming language clause always contains program statements that execute if the condition is true, while a filter clause contains outcomes whose nature depends on what kind of dialogue component contains the filter pattern. Second, in a general-purpose programming language the specific “clause container” defines the semantics of the relationship between the clauses it contains. For example, in the C-language, each case clause is independent and the conditions in one case clause do not affect the other clauses. In Ada, each else-if clause has a “short-circuit” semantics so that the clause is not evaluated if the condition in the previous clause had value true. With filters, there is a single syntactic filter construct that contains a list of clauses, but the semantics of the relationship between clauses in the filter depends on the dialogue component that contains the filter.

A filter pattern can be used in four different ways and the outcomes and clause sematics are different, depending on how it is used:

1. In a decision, only the first clause that evaluates to true is considered and each clause contains exactly one outcome that is the first component of a topic or another decision (that can be the first component of another pattern)
2. In an *agent remark decision*, only the first clause that evaluates to true is considered and each clause contains exactly one outcome that is an agent remark.
3. In a *user choice filter,* each clause is evaluated independently and can contain zero or more outcomes that are user choices,
4. In a *global user remark filter,* each clause is evaluated independently and each clause contains exactly one outcome that is text,

Each filter pattern can be used for any of these four purposes by placing it in the appropriate dialogue component. For example, if an *Ability HighLow* filter was used in a user choice filter, each clause could contain multiple outcomes (user choices) and all of the user choices in any clause that evaluated to true would appear in a dialogue. If the same filter was used in an agent remark decision, only one outcome (agent remark) could appear in each clause and only the first clause to evaluate to true will have its agent remark displayed. Since the clauses in an *Ability HighLow* filter are mutually exclusive, the second difference is not important. However, for filters whose clauses are not mutually exclusive, the context-dependent clause evaluation semantics are important. Topic and agent remark decisions contain the word *decision* since each clause must have exactly one outcome, while user choice filters contain the word *filter* since each clause can have zero or more outcomes. Global user remark filters will be discussed later.

Filter patterns can also contain other filter patterns (nesting). In this case, an outcome is only used if all of the conditions of all of the filters containing it evaluate to true. A filter pattern can contain a compound condition such as the *Insight* pattern in which PC *Intelligence* must be at least as high as one threshold option and the PC *Wisdom* must be at least as high as another threshold option. This pattern is used to provide additional dialogue choices to insightful PCs. For example, in Chapter 1 of the NWN official campaign story there are 126 user vertices where an *Insight* pattern could be used to filter the vertices.

In general-purpose programming languages a Boolean expression or a clause is usually the unit of re-use that can appear in different syntactic structures and have different semantics based on its container. For example, in the C-language the clause in an if-statement and while statement has the same syntax and the semantics is different based on the container. In our dialogue model, a filter pattern with its options, clauses and outcomes is a re-usable component at a higher level of abstraction than a single clause. This means that a single library of filters can be used for all three purposes, rather than supporting a different library for user choice filters, decisions and agent remark decisions.

8.1 Decisions and Decision Icons

An author often wants to select a dialogue path based on the game state. Scripts can be used to select between vertices. For example, in Fig. 1 and Fig. 2, the sibling agent vertices A to D have scripts and the dialogue path is selected by evaluating the scripts and selecting the first vertex whose script evaluates to true. For example, Fig. 3 shows the NWScript scripts for two of these vertices (B and D). In Fig. 1 and Fig. 2, agent vertex B should only appear if the Docks quest of the story has not been completed before the dialogue takes place and if the dialogue is with the another NPC instead of with the PC. This dialogue control information is missing from the vertex-oriented dialogue graph representations in Fig. 1 and Fig. 2. In the dialogue pattern model, a decision about dialogue paths can be encapsulated explicitly using a *decision* that is constructed from a filter pattern to select among topics and other decisions. For example, in Fig. 8, a *Quest completed & NPC-NPC dialogue* filter pattern is used in a decision to select among four topics. If the PC has completed the *Docks* quest in the game and the NPC is talking to another NPC (with the PC within hearing range) then outcome 1 is selected so that the *Safe* topic is selected. In the most important case, labeled *otherwise* (*The Docs quest is not completed and the NPC is talking to the PC*), outcome 4 is selected and topic *I want to work* is used. When a filter is used in a decision, the graphical user interface ensures that each clause contains exactly one outcome and the condition of the last filter clause is removed and replaced by an always-true condition that is labeled *otherwise*. The generated code selects the outcome of the first clause to evaluate to true and at least one clause must evaluate to true since the last clause has an always-true condition.

It is possible for two user choices to link to the same decision. For example, an author could unshare user choice JH-5 in the dialogue of Fig. 8, and set the user link to the decision *Dock quest completed & NPC-NPC dialogue.* In fact, this would make a silly dialogue, but in general, this mechanism is useful. A *decision icon* would appear, similar to the topic icon in Fig. 11, except that the name of the topic would be replaced by the name of the decision (*Dock quest completed & NPC-NPC dialogue*) and a yellow rectangle (decision icon) would be used instead of the yellow rounded rectangle (topic icon). If this user choice was then shared (say as MH-5), then the link icon for this shared user choice would have an arrowhead in a yellow rectangle (shared decision icon), instead of the yellow rounded rectangle (shared topic icon).

8.2 Agent Remark Decisions

Sometimes multiple agent remarks result in the same subsequent dialogue. For example, it is common for an agent to have a different remark depending on the PC's *Charisma*, but for the user choices that follow to be independent of the agent remark. This is the case in Fig. 1 and Fig. 2, where agent vertices D, E and F have user choices, D-1, E-1 and F-1respectively, each with blank user remark and a common user link, agent remark G. In such cases, we can use a filter pattern in an *agent remark decision* to combine the agent remarks into an *agent menu* and place the agent menu into a single exchange. Fig. 8 shows how agent remarks D, E and F can be combined into a single exchange, using an agent remark decision. When a filter pattern is used as an agent remark decision, the graphical user interface ensures that exactly one outcome is placed in each filter clause so that the agent has exactly one remark that begins the exchange. The last clause is also changed to an otherwise clause with a condition that is always-true. This ensures that the agent always has a remark to select.

8.3 User Choice Filters

When constructing user choices, an author may want certain user choices to be available only when certain conditions are met. In the dialogue pattern model, user choices can be filtered by instantiating a *filter pattern* in the user menu of an exchange. In this case each filter pattern clause will contain a condition and room for the author to insert a list of user remarks. Two of the most common uses of filter patterns in the NWN campaign dialogues are to enable or disable a single user choice (e.g. the *Insight* pattern) or to enable exactly one of two user choices (e.g. *Ability HighLow* pattern). When a filter is used to enable exactly one of a pair of user choices and both of their user links lead to the same next topic, we call the filtered user choices, *degenerate user choices* and this pair of user choices are placed in a user choice group. Exchange H in Fig. 8 contains a pair of degenerate user choices, H-3 and H-4, along with a pair of filtered user choices that are not degenerate, H-1 and H-2 and one user choice that is not filtered, H-5.

User choice filters can appear in shared user menus, such as the shared user menu in exchanges H and N in Fig. 8. In this case, the filter itself is shared so that changing the filter conditions or the outcomes in one instance, changes them in all instances. For example, if user choice H-5 is moved into one of the filter clauses in exchange H then user choice NH-5 would be automatically moved into the same filter clause in exchange N. However, shared user choices do not share dialogue control (only user remarks and user links), so moving H-5 or NH-5 into a filter clause would not affect the dialogue control for user choice JH-5 or MH-5. This semantics is different than the semantics used by the Aurora conversation editor, where user link lines share scripts with the user line to which they correspond. For example, if the script on user line H-3 of Fig. 2 is changed, the script on user link line MH-3 is automatically changed. In the pattern model of Fig. 8, if the script on user choice H-3 is changed by moving it from its current place in the filter, there is no effect on the dynamic dialogue control (scripts) for user choice MH-3, even though it is a shared user choice, since only the user choice is shared, not the control which is part of its non-shared user menu.

In the pattern model, an author has a choice whether to share dialogue control by using shared user menus or to share only the user choice (user remark and user link) by using shared user choices. In addition to providing sharing at a higher level of abstraction, this semantics can have a performance advantage. In the *Docks* conversation shown in Fig. 8, no dialogue control scripts are generated for user choice MH-3, since there are no filters in the user menu of exchange M, even though user choice MH-3 is shared with user choice H-3, which has dialogue control scripts generated by the filter that contains it. However, in the Aurora conversation model, a script on line MH-3 is executed at run-time. This script is not necessary since any dialogue path to this user choice must pass through either user choice H-1 or its shared user choice NH-1, which are already guarded by a high *Intelligence* filter.

8.4 Global User Remark Filters

A *degenerate user choice* occurs when a filter is used to enable exactly one of a pair of user choices and both of their user links lead to the same next topic. User choice groups provide one mechanism to model this phenomenon and a single link for the group replaces individual links for group members. In Chapter 1 of the NWN official campaign story, of the 5,226 instances of the *Ability HighLow* filter pattern that could be used, 3,956 of these would result in degenerate user choices. Since degenerate user choices are so common, our dialogue pattern model has an alternate mechanism that further exploits them. If a single *global user remark filter* is defined (at the root component), then each user remark in the pattern graph represents an instance of this global user remark filter. The author provides one user text for each filter clause in each user remark. In NWN, the most common example of this situation occurs with *Intelligence HighLow* filter patterns, where for some conversations, the author provides two user texts for each user remark, one for high intelligence and one for low intelligence. The advantage of using this global filter is that the variation is localized to text entry and does not affect the structure of the dialogue. Fig. 12 shows the pattern graph of the Docks conversation, when a global *Intelligence HighLow* user remark filter is used. User and agent remark labels from Fig. 2 have been added to Fig. 12 to illustrate the correspondence and the layout has been changed to fit on one page for this paper. The layout normally has no crossing lines, but takes up more screen real estate.

The user choice filter in exchange H disappears since it is subsumed by the global user remark filter. However, one new decision appears in the graph since in addition to most of the user remarks depending on the PC's intelligence, there is also one structural path choice (topic *Others know more* or topic *Won't explain*) that depends on the (PC's) intelligence. Note that an agent remark filter could be used instead of this decision if the *Others know more* topic contained only exchange M, since the user menus for exchange M and J are the same. However, the *Others know more* topic also contains exchange I that is collapsed in Fig. 12 and this extra exchange precludes the use of an agent remark filter to simply pick between agent remark M and J.

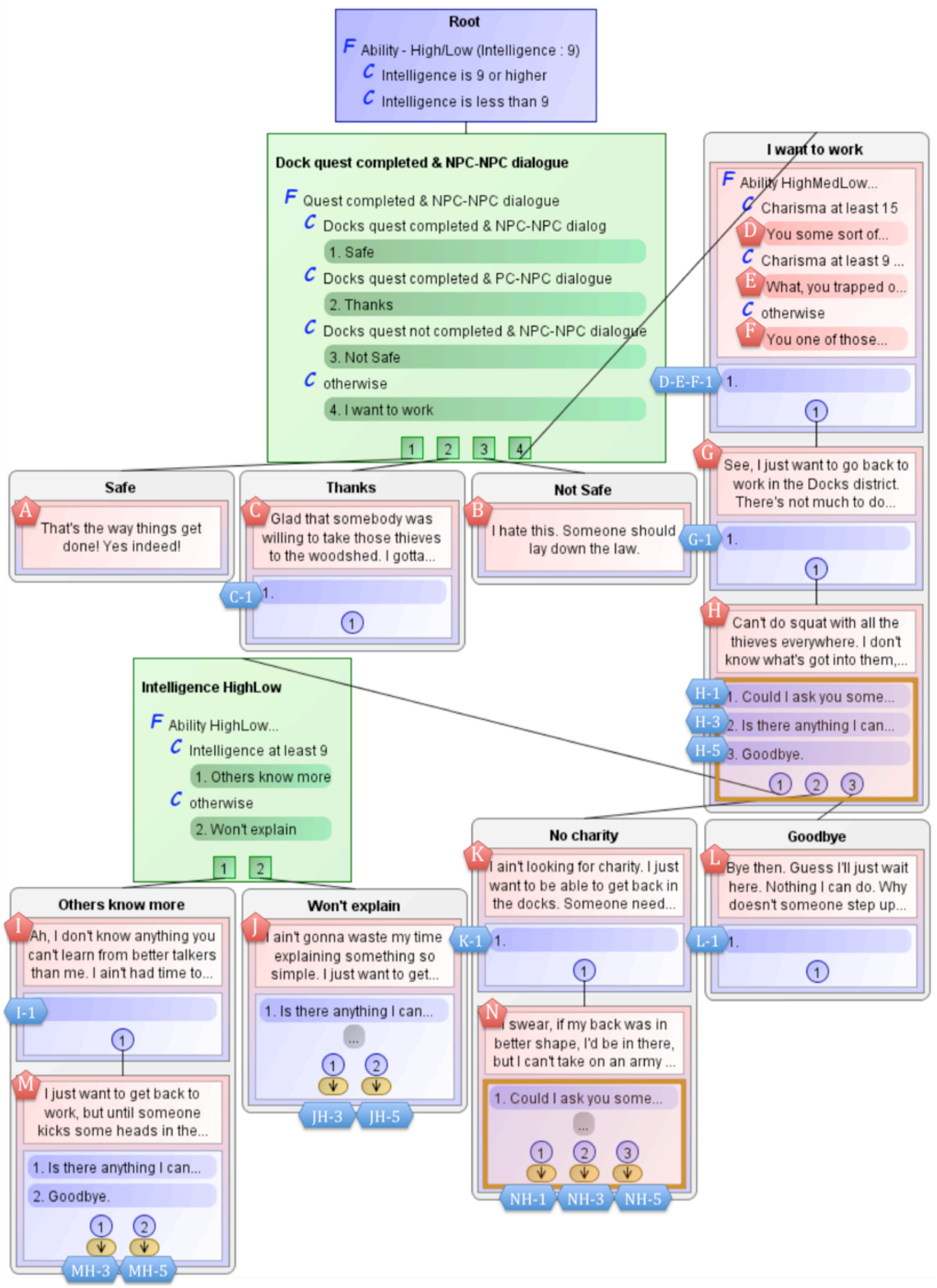
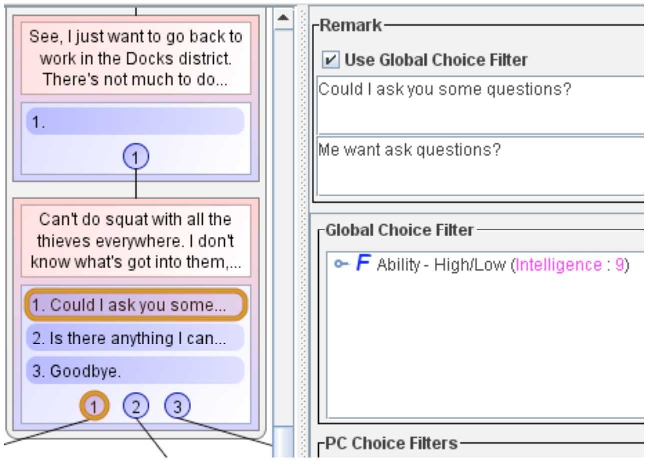

Root
F Ability - High/Low (Intelligence : 9)
C Intelligence is 9 or higher
C Intelligence is less than 9
Dock quest completed & NPC-NPC dialogue
F Quest completed & NPC-NPC dialogue
C Docks quest completed & NPC-NPC dialog
1. Safe
C Docks quest completed & PC-NPC dialogue
2. Thanks
C Docks quest not completed & NPC-NPC dialogue
3. Not Safe
C otherwise
4. I want to work
I want to work
F Ability HighMedLow...
C Charisma at least 15
D You some sort of...
C Charisma at least 9 ...
E What, you trapped o...
C otherwise
F You one of those...
D-E-F-1 1.
G See, I just want to go back to work in the Docks district. There's not much to do...
G-1 1.
H Can't do squat with all the thieves everywhere. I don't know what's got into them,...
H-1 1. Could I ask you some...
H-3 2. Is there anything I can...
H-5 3. Goodbye.
Safe
A That's the way things get done! Yes indeed!
Thanks
C Glad that somebody was willing to take those thieves to the woodshed. I gotta...
C-1 1.
Not Safe
B I hate this. Someone should lay down the law.
Intelligence HighLow
F Ability HighLow...
C Intelligence at least 9
1. Others know more
C otherwise
2. Won't explain
No charity
K I ain't looking for charity. I just want to be able to get back in the docks. Someone need...
K-1 1.
N I swear, if my back was in better shape, I'd be in there, but I can't take on an army ...
1. Could I ask you some...
NH-1 NH-3 NH-5
Goodbye
L Bye then. Guess I'll just wait here. Nothing I can do. Why doesn't someone step up...
L-1 1.
Others know more
I Ah, I don't know anything you can't learn from better talkers than me. I ain't had time to...
I-1
M I just want to get back to work, but until someone kicks some heads in the...
1. Is there anything I can...
2. Goodbye.
MH-3 MH-5
Won't explain
J I ain't gonna waste my time explaining something so simple. I just want to get...
1. Is there anything I can...
JH-3 JH-5

Fig. 12. A pattern graph for the *Docks* conversation that exploits a global user remark filter

The semantics of the filter clauses in a global user remark filter is that each clause is evaluated independently and each clause contains exactly one outcome that is text. In most cases, the clauses should be constructed to be mutually exclusive so that exactly one user text is displayed. For example, this is the case for an *Ability HighLow* filter. However, if more than one user text is selected by a global user remark filter, they will both appear in the dialogue and share a link. If for a particular user remark, the author wants to ignore the global user remark filter and have a single user text, then a checkbox is used to ignore the global user remark filter for that user remark. The global user remark filter is associated with the root node. Fig. 12 shows the global user remark filter in the root node and Fig. 13 shows the text entry dialogue that can be used to enter the user text for each clause of each user remark. One sub-pane appears for each clause in the global choice filter.


See, I just want to go back to work in the Docks district. There's not much to do...

1.

Can't do squat with all the thieves everywhere. I don't know what's got into them,...

1. Could I ask you some...
2. Is there anything I can...
3. Goodbye.

Remark

[✓] Use Global Choice Filter

Could I ask you some questions?

Me want ask questions?

Global Choice Filter

F Ability - High/Low (Intelligence : 9)

PC Choice Filters

Fig. 13. Entering the user text for a global user remark filter

In small conversations, there is not much difference in the complexity of dialogues constructed with or without global user remark filters. However, we will see in the case study that global user remark filters can have a significant impact on larger conversations, when most user remarks are duplicated for a single trait like PC intelligence.

9. Case study

To evaluate the effectiveness of the dialogue pattern model, five representative conversations from Chapter 1 of the NWN official campaign were selected for a case study. The conversations were expressed both as a textual remark graph using the Aurora conversation model and as a graph using the dialogue pattern model. The models were then directly evaluated by applying eight metrics to the two different graph representations. Each metric was designed to measure an aspect of the *simplicity* of the graphs. The representation with the lowest simplicity metric scores should be easier to use. The disadvantage of this approach is that we must deduce ease-of-use from simplicity metrics. Fig. 14 is a histogram of the lengths of all conversations in Chapter 1 of the NWN official campaign story. The lengths of the five conversations used in this case study are highlighted so that you can see where they fall in the distribution. We selected these conversations to specifically cover the whole distribution.

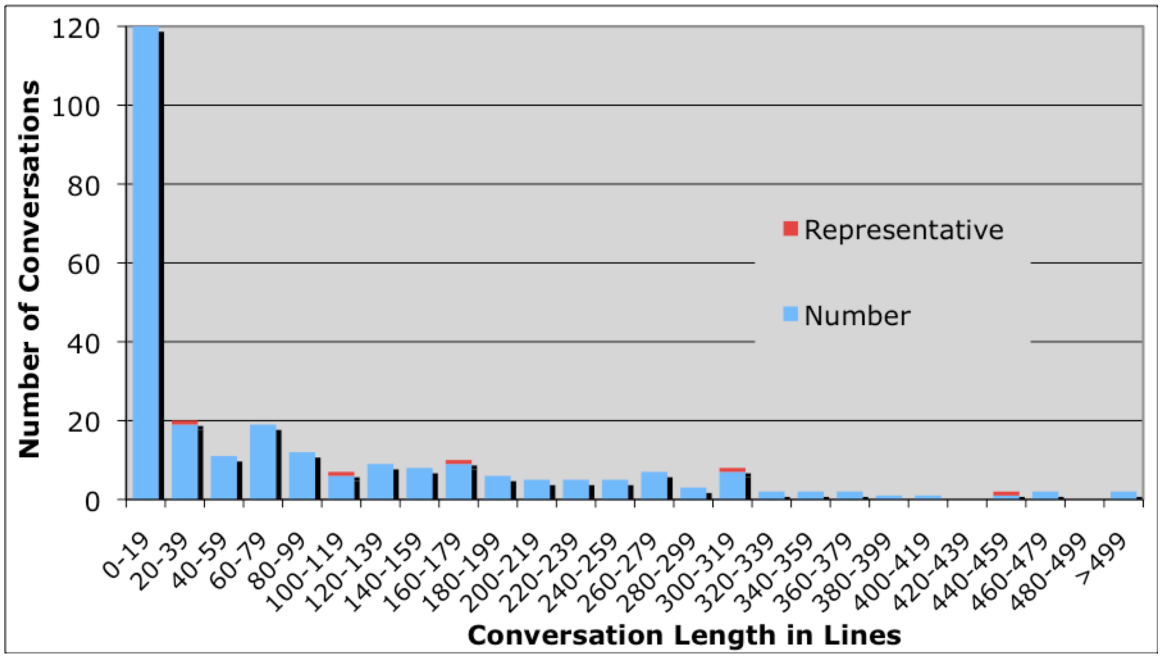


Fig. 14 A histogram of all conversations from Chapter 1 of the NWN official campaign story based on the number of lines in each conversation. The five conversations used in the case study are highlighted (dark).

An alternative evaluation would be a user study that surveys a group of authors who would directly rate the ease-of-use of two different tools, where each tool implements one of the models. The advantage of this approach would be that we would not have to deduce ease-of-use from simplicity metrics. The disadvantage is that such a study would rely on the particular implementation of the two models. A poorly implemented tool could mask a good model and a great tool could cause the author to overlook some problems with a model.

We decided to start with a case study as the best way to directly evaluate the models, without bias from tools that implement the models. We plan on conducting a user study similar to the studies we have conducted on other components of ScriptEase [22] to ensure that our simplicity metrics translate to ease-of-use, at least in the game dialogue-authoring domain. Our current case study uses seven simplicity metrics: structural, topological, remark, indirection, control, structure-operational and control-operational simplicity. The three metrics that contain the word, control, measure control simplicity and the other four metrics measure structural simplicity. The simplicity metrics are denoted with a subscript R for the remark model (Aurora in our case study), and subscript P for the dialogue pattern model. For example, structural simplicity for a conversation in the remark model would be structural_R(conversation).

9.1 The Simplicity Metrics

Table 1 can be used to quickly determine what graph components are counted in each metric, for each model. A detailed description of each metric for the two models follows. We use the now familiar *Docks* conversation as an example to illustrate the metrics. We have picked this very simple conversation for ease of exposition. However, this conversation is quite simple compared to many of the conversations found in the NWN campaign story, so the utility of the pattern model is not as obvious as when the metrics are applied to the other larger representative conversations selected for the case study. The *Docks* conversation is the leftmost highlighted conversation in the histogram of Fig. 14, with 36 lines (20-39).

Table 1 The simplicity metrics for the remark and dialogue graphs. The component names are: agent lines (AL), user lines (UL), user lines with text (ULWT) link lines (LL), agent remarks (AR), user links (ULK), user text (UT), link icons (LI), topic icons (TI), decisions (D), decision icons (DI), non-link lines with a tilde indicating an attached script (NLTL), filters (F), non-otherwise filter clauses (FC), and scripts (S).

Metric	Remark dialogue graph	Pattern dialogue graph
structural	AL+UL+LL	AR+UC+TI+D+DI
topological	AL+UL+LL (collapsed)	AR+UC+TI+D+DI(collapsed)
remark	AL + ULWT	AR + unique UT
indirection	LL	LI + TI + DI
control	NLTL	FC or F
structure-operational	2 * (AL + UL + LL)	Count operations (see text)
control-operational	NLTL + unique S	F + unique F

Structural simplicity measures the number of visible structural components when a conversation is fully expanded and dialogue control components are ignored. In the case of the remark conversation model, this includes all agent lines, user lines, and link lines. Therefore, the structural simplicity is just the length of the conversation, measured in total lines, as shown in Fig. 14.

In the case of dialogue patterns, this metric includes all agent remarks, user choices, topic icons, decisions and decision icons in a graph in which no topics are collapsed. This metric does not include filters since they are dialogue control components, not structural components. For example, the decision pattern below the root in Fig. 12 is ignored and the author can think of the root as connecting to all four outcomes of this decision pattern. This is analogous to what happens in a remark graph where the scripts are ignored. Exchanges are not counted separately. An exchange that contains a single agent remark is synonymous with the agent remark it contains. An exchange that contains more than one agent remark (exchange D-E-F in Fig. 8) is synonymous with its agent remark decision, which is only important for dialogue control. However, in this case, we count each of the multiple agent remarks in the exchange. We count user choices by counting circle icons rather than counting user remarks or user links, since some user choices have blank user remarks (C-1, D-E-F-1 G-1, I-1, K-1 and L-1 and since user choice groups share links (H-3, H4 and NH-3, NH-4). Counting user choices yields the highest number (worst score) so it is the most pessimistic. We count dimmed user choices since they appear in the graph, even though they should be ignored by the author. However, we do not count each user text of user remarks in global user remark filters since they do not appear in the graph (they are not structural). To eliminate layout dependencies, we make the pessimistic assumption that each topic is the target of exactly one user link and all other user links to that topic use topic icons. This means that in practice, the structural simplicity of a pattern graph will often be lower than our metric indicates, since it is common for nearby user links to link directly to the same topic. For example, in Fig. 11, the link to the *Goodbye* topic icon could be replaced by a direct link to the *Goodbye* topic without making the graph in Fig. 8 non-planar.

As an example of structural simplicity, consider the Docks conversation represented in Fig. 2, Fig. 8 and Fig. 12. In the remark model, there are 14 agent lines (A – N), 12 user lines (C-1, D-1, G-1, H-1, I-1, H-2, H-3, K-1, H-4, H-5, E-1 and F-1) and 10 link lines (MH-1, MH-5, JH-4, JH-5, NH-1, NH-2, NH-5, OK, PG, QG). Therefore, we have structural_R(Docks) =

14 + 12 + 10 = 36. The author views 36 total components when the conversation is fully expanded. In the pattern graph of Fig. 8, there are 14 agent remarks (A – N), 20 user choices (C-1, D-E-F-1, G-1, H-1, H-2, H-3, H-4, H-5, I-1, JH-4, JH-5, K-1, L-1, MH-3, MH-5, NH-1, NH-2, NH-3, NH-4, NH-5), 0 topic icons, 1 decision and 0 decision icons. Therefore, we have $\text{structural}_P(\text{Docks}) = 14 + 20 + 1 = 35$. For this example, the pattern model has a structural simplicity that is 97% of the structural simplicity of the remark model. As shown in the case study results, as the graphs get larger this advantage remains in the 86% to 94% range. Alternately, we could use a global user remark filter and compute the structural simplicity from the pattern graph shown in Fig. 12. In this case there are the same 14 agent remarks (A - N), 16 user choices (C-1, D-E-F-1, G-1, H-1, H-3, H-5, I-1, JH-3, JH-5, K-1, L-1, MH-3, MH-5, NH-1, NH-3, NH-5), 0 topic icons, 2 decisions and 0 decision icons. Therefore, we have $\text{structural}_P(\text{Docks}) = 14 + 16 + 2 = 32$. In this case, the structural simplicity is 89% as high as for the remark graph. This is an example of a conversation where, there is not much difference in the complexity of dialogues constructed with or without global user remark filters. However, the case study shows that for larger graphs, this advantage is in the 60% to 65% range.

Topological simplicity measures the number of visible components when all components that are unnecessary for the author to understand the branching structure of the conversation are hidden. In the case of the remark model, the only vertices that can be collapsed are those that form a linear chain that terminate in an explicit [END DIALOGUE] vertex (chain C, C-1 in Fig. 2) or are followed by an implicit end dialogue vertex (chain H-5, L in Fig. 2). In Fig. 2, these linear chains result in a reduction of 2 lines (C-1 and L). Therefore, $\text{topological}_R(\text{Docks}) = 36 – 2 = 34$. With the pattern model, there is potential for more abstraction, since all of a topic's inner exchanges can be collapsed. This is equivalent to collapsing any linear chain of vertices that starts with an agent vertex and ends with an agent vertex that has zero or more user vertices following it. It is more general than the remark model in two important ways. First, the chain does not have to end in a leaf node (explicit or implicit). Second, the last item in the chain can have multiple user vertices. It is less general than the remark model in one way – it cannot start with a user vertex, such as the chain (H-5, L) in Fig. 2. Comparing the three collapsed topics shown in Fig. 10 with their uncollapsed versions in Fig. 8, we exclude 6 agent remarks (I, K, D, E, F, and G) and 4 user choices (I-1, K-1, DEF-1 and G-1) contained in the three collapsed topics, so $\text{topological}_P(\text{Docks}) = 35 – 10 = 25$. This value is 74% of the topological simplicity of the remark graph. As shown in the case study results, this percentage varies from 77% to 104% for other graphs. However, if instead, we use the global user remark filter from Fig. 12, we obtain the same reduction of 6 agent remarks and 4 user choices so $\text{topological}_P(\text{Docks}) = 32 – 10 = 22$, which is 65% as high as the topological simplicity of the remark graph. This advantage varies from 56% to 70% for other conversations in the case study.

Remark simplicity measures the number of remarks the author enters as text when building the conversation. This metric directly measures the number of remarks the author must write, independent of the number of components in the conversation. For the remark conversation model, this metric is the sum of agent lines and all user lines that actually contain text (we do not count the [CONTINUE] or [END DIALOGUE] lines). Link lines are ignored since they have no settable remark text. For example, the Docks graph in Fig. 2 has 14 agent lines (A – N) and 5 user lines (H-1 to H-5) that have text. Therefore $\text{remark}_R(\text{Docks}) = 14 + 5 = 19$. In the case of the pattern model, remark simplicity is the number of agent remarks plus the number of unique user texts. If no global user remark filter is used, the number of unique user texts is the number of unique user remarks. However, if a global user remark filter is used, all of the user remarks that use it will have a number of user texts equal to the number of clauses in the filter. Each shared user remark has its user texts counted once.

The Docks dialogue pattern graph in Fig. 8 has 14 agent remarks (A – N) plus 5 unique user remarks H-1 - H-5. All other user remarks (JH-4, JH-5, MH-3, MH5, NH-1, NH-2, NH-3, NH-4, NH-5) are shared versions of these 5 user remarks. Therefore $\text{remark}_P(\text{Docks}) = 14 +5 = 19$. Alternately, if a global user remark filter is used (Fig. 12), there are 14 agent remarks (A – N) plus 2 unique user remarks H-1 and H-3 that each have two user texts (high intelligence and low intelligence) plus one user remark, H-5, that does not user the global filter so it has a single user text. All other user remarks (JH-3, JH-5, MH-3, MH-5, NH-1, NH-3, NH-5) are shared versions of these 3 user remarks. Therefore $\text{remark}_P(\text{Docks}) = 14 + 2*2 + 1 = 19$. Without shared user menus and shared user remarks, the remark simplicity of a pattern graph would be higher. For example, without this sharing, the remark simplicity of the pattern graph for the Docks conversation would be 25 instead of 19. With the sharing the remark simplicity for remark and pattern graphs is always the same unless an author neglects to share a remark when it could be shared.

Indirection simplicity measures the number of points in the conversation where the dialogue graph is terminated with a leaf node that acts as a placeholder for another component. This metric measures the number of context switches necessary when trying to view all indirect relationships in the graph, where the author must follow a link to its target. In the case of the Aurora conversation model, this metric is the number of link nodes. For example, the Aurora Docks remark graph has 10 link lines. Therefore $\text{indirection}_R(\text{Docks}) = 10$. In the case of the dialogue pattern model, this is the number of undimmed link icons plus the number of topic icons plus the number of decision icons. The *Docks* pattern graphs in Fig. 8 and Fig. 12 (which uses a global user remark filter), each have 7 undimmed link icons, 0 topic icons and 0 decision icons. Therefore $\text{indirection}_P(\text{Docks}) = 7$. This reduction in indirections together with a refocusing mechanism that replaces icons with direct targets at the expense of icons elsewhere in the graph significantly reduces the negative effects of context switches. This reduction to 70% of the remark indirection is a characteristic of the small dialogue graph. For the other four conversations, the advantage is much better at 32% to 45%.

Control simplicity measures the number of dialogue control components required by the model. Although the remark graph does not display its dialogue control components, filter scripts (shown in Fig. 3) must be attached to remark vertices to control whether the remarks are displayed or not. For a remark graph, the detail-control simplicity is defined as the number of non-link remark vertices that require scripts. Link lines are not counted since they simply contain a reference to a line that has a script and the author does not need to reconsider the control that script provides in the link line. In Fig. 2, lines requiring a filter script are marked with a square icon containing a tilde (~). For example, from Fig. 2, the non-link tilde lines are A, B, C, D, H-1, H-2, H-3, H-4, and E, so the detail-control simplicity of the Docks conversation in the remark graph is control_R(Docks) = 9. The remark graph relies on some implicit control information that is not represented by a ~ icon in the graph. If an agent line has a filter script, then the script affects all of its siblings, since only the first agent line whose script evaluates to true is displayed. For example, in the remark graph in Fig. 2, there is implicit control on line F, since it will only appear if the filter scripts on lines A, B, C, D, and E, evaluate to false. In fact, any script attached to the last agent sibling is ignored, since there must always be exactly one agent remark at runtime and if the filter scripts on all previous siblings evaluate to false, the last agent sibling remark must be displayed. Therefore, even if a script is added to the last agent sibling, it is not marked with a ~ icon and it is ignored at run-time. For example, line F of the Docks conversation shown in Fig. 2, actually has a filter script that is ignored. We do not count this ignored script even if a script is placed on this line, since the script is never used.

In pattern graphs, we have two choices. We can either count clauses or count filters, so we define two sub-metrics, *detail-control* and *abstract-control*. If we count clauses, we mirror our decision not to count the implicit control on the last sibling in a set of scripted sibling agent lines, by ignoring *otherwise* clauses. For example, the Docks pattern graph in Fig. 8 has 3 non-otherwise clauses in the decision (*Dock quest completed & NPC-NPC dialogue*), 2 non-otherwise clauses in the agent remark decision of exchange D-E-F, and 2 clauses from the user choice filter in the shared user menu of exchanges H and N, so detail-control_P(Docks) = 7. The detail-control simplicity of the pattern model is 78% as high as the control simplicity of the remark model and this simplification varies widely in the larger graphs in the case study (44% to 93%). However, it can be argued that control simplicity in the pattern model should be measured at a higher level of abstraction. We could count only the number of filter patterns, not the number of clauses in the patterns, since each filter pattern is a reusable component that abstracts a single concept. When this metric is applied to the Docks pattern graph, the value is $\text{abstract-control}_P$(Docks) = 3 which is 33% of the control simplicity of the remark model and for the other conversations used in the case study, this advantage varies from 19% to 37%. If we apply the control metric to the pattern graph of Fig. 12, which uses a global user remark filter, we obtain exactly the same counts. The user choice filter that appears in Fig. 8 is replaced by the global user choice filter that appears in the root. However, as we shall see in the case study, there are some conversations where using a global user remark filter significantly reduces the control simplicity with very small ratios to remark graphs ranging from 33% to 44% for detail control and 14% to 34% for abstract control.

Structure-operational simplicity measures the minimum number of operations the author must perform to construct the structure of the dialogue graph. For simplicity, all operations are assigned a cost of 1 unit. This metric estimates the minimum amount of work required to construct the structure of the conversation using the model, assuming no errors are made and the dialogue is known in advance, so that the number of operations can be minimized. For remark graphs, the operations used are: *add vertex*, *copy vertex*, *paste as link* and *enter remark*. Each agent vertex and user vertex requires two operations, *add vertex* and *enter remark*. Even blank remarks require the user to use an edit field to erase an <*insert text here*> placeholder. Each link line also requires two operations, one *copy vertex* and one *paste as link*. Therefore, the structure-operational simplicity of a remark graph is always two times the structure simplicity. The structure-operational simplicity of the Docks remark dialogue graph shown in Fig. 2 is $\text{structure-operational}_R$(Docks) = 72. For pattern graphs, the operations used in computing this metric are: *add topic*, *enter agent remark*, *add user choice*, *remove user choice*, *enter user text*, *insert exchange*, *link*, *copy user menu*, *share user menu*, *copy user choice*, *share user choice*, *disable user choice* and *add decision*. Some operations implicitly contain other operations. For example, when an *add topic* is performed on a user choice, a topic is created that contains one exchange and a link is created from the user choice to the topic. The exchange contains a blank agent remark, a blank user remark and a user link. The Docks pattern graph shown in Fig. 8, required: 8 *add topic*, 14 *enter agent remark*, 4 *add user choice*, 2 *remove user choice*, 5 *enter user remark*, 4 *insert exchange*, 0 *link*, 1 *copy user menu*, 1 *share user menu*, 3 *copy user choice*, 4 *share user choice*, 2 *disable user choice* and 1 *add decision* to create the structure. Note that no link operations are necessary since when a *share user menu* or *share user choice* is performed, the user link is automatically set to the copied user link of the user choice. Therefore, structure-operational*P*(Docks) = 49. This metric is 68% as large as the remark graph. However, the fact that there are 12 different operations as opposed to only 4 different operations for the remark graph could make graph construction more difficult for a novice user. Nevertheless, after learning the operations, there are fewer to perform to construct each graph. If instead, we use a global user remark filter (Fig. 12), we require: 8 *add topic*, 14 *enter agent remark*, 2 *add user choice*, 2 *remove user choice*, 5 *enter user text*, 4 *insert exchange*, 0 *link*, 2 *copy user menu*, 2 *share user menu*, 2 *copy user choice*, 2 *share user choice*, 0 *disable user choice* and 2 *add decision* to create the structure. Therefore, with a global user remark filter, structure-operational*P*(Docks) = 45, which is 63% as large as the remark graph. Similar reductions exist for the other conversations used in the case study.

Control-operational simplicity measures the minimum number of operations the author must perform to construct the control for a dialogue graph. Again, all operations are assigned a cost of 1 unit. This metric estimates the minimum amount of work required to add control information to the conversation, assuming no errors are made, the dialogue is known in advance, so that the number of operations can be minimized and the scripts are already written but not attached. For remark graphs, the two operations are *pick script* (selecting or adapting an appropriate script) and *attach script* (attaching the script to a vertex of the graph). The actual construction of the script is ignored, even though it requires considerable effort to construct these scripts, since in theory, scripts could be reused or written by an individual other than the one creating the dialogue. However, picking a script requires the author to understand the purpose of the script and may require the author to specify some adaptations to an existing script. In the remark graph of Fig. 2, lines with scripts are marked with a tilde. However, link lines are simply copies of existing lines, so no *attach script* operation is required. The structure operations *copy vertex* and *paste as link* are sufficient to transfer the script. Therefore, we count the number of non-link tilde lines plus the number of unique scripts so that control-operational$_R$(Docks) = 7 + 9 = 16. For pattern graphs, we have the operations: *add filter*, *replace filter* and *set filter option*. Note that the structural operations, *add user choice*, *copy user menu*, *share user menu* and *disable user choice*, also manipulate control information along with the structure, but they are counted with the structure operations so they are not counted again in this metric. In addition to these operations, we count the number of unique filters that were used, since the author must understand them. The Docks pattern graph shown in Fig. 8, required: 2 *replace filters*, 1 *add filter* and 1 *set filter option*, for a total of 4 operations to attach the control. The *Quest completed & NPC-NPC dialogue* filter required an option (*Docks*), while the *Ability HighMedLow* used three default parameter values (*Charisma*, *9* and *15*) and the *Ability HighLow* filter used two default option values (*Intelligence*, *9*). In addition, 3 different filter patterns were selected for use. Therefore, control-operational$_P$(Docks) = 4 + 3 = 7. The pattern model scores 44% of the control-operational metric compared to the remark model. If we use a global user remark filter (Fig. 12), there are 4 *replace filters* (including the global one), 0 *add filter*, and 1 *set filter option*, and 3 different filter patterns (an *Ability HighLow* filter is used both as a global user remark filter and as a decision filter) for a total of 8 operations. This reduces the advantage of the pattern graph by increasing the simplicity score to 50% of the remark graph score. However, as shown in the case study, the global user remark filter lowers the simplicity metric score of larger conversations.

9.2 Results

Five representative conversations were used in this case study, *Docks*, *Siri, Nurse*, *Tani*, and *Bertrand*. These five conversations are a good representation of conversations of different sizes that appear in Chapter 1 of the official NWN campaign story, based on the histogram in Fig. 14. The *Bertrand* conversation is one of the most complex conversations. This conversation is highly dynamic and uses many control patterns.

The simplicity metric scores are shown in Table 2. Each conversation has three rows, one for the remark model (R), one for the pattern model (P) without using a global user remark filter and one uses a global intelligence user remark filter. Each pattern row also contains the ratio (P/R). The averages of the scores and the ratios are shown in the last three rows.

Table 2 The simplicity metric scores of remark graphs (R) and pattern graphs (P) for five representative dialogues from the NWN official campaign game adventure. The metrics are: structural (struct), topological (topo), remark (remark), indirection (indir), detail and abstract control (d-cont & a-cont), structural-operational (s-op) and control operational (c-op). There are two lines for each P metric, one that does not use a global user remark filter and one that does and each entry contains the metric value and its ratio to the R metric value.

	struct	topo	remark	indir	d-cont	a-cont	s-op	c-op
Docks$_R$	36	34	19	10	9	9	72	16
Docks$_P$	35/.97	25/.74	19/1.0	7/.70	7/.78	3/.33	49/.68	7/.44
Docks$_P$-global	32/.89	22/.65	19/1.0	7/.70	7/.78	3/.33	45/.63	8/.50
Siri$_R$	100	92	50	50	36	36	200	49
Siri$_P$	96/.96	96/1.04	49/.98	16/.32	16/.44	7/.19	113/.57	9/.18
Siri$_P$-global	64/.64	64/.70	49/.98	16/.32	12/.33	5/.14	93/.47	7/.14
Nurse$_R$	176	176	70	89	38	38	352	48
Nurse$_P$	152/.86	136/.77	70/1.0	40/.45	25/.66	13/.34	200/.57	16/.33
Nurse$_P$-global	115/.65	99/.56	70/1.0	36/.40	14/.37	9/.24	162/.46	13/.27
Tani$_R$	308	295	112	172	72	72	616	103
Tani$_P$	289/.94	269/.91	112/1.0	68/.40	55/.76	23/.32	329/.53	33/.32
Tani$_P$-global	201/.65	181/.61	112/1.0	68/.40	32/.44	18/.25	262/.43	28/.27
Bertrand$_R$	450	432	151	277	97	97	900	127
Bertrand$_P$	421/.94	391/.91	150/.99	95/.34	90/.93	36/.37	495/.55	47/.37
Bertrand$_P$-global	271/.60	241/.56	150/.99	92/.33	47/.48	33/.34	381/.42	44/.35
Average$_R$	214	206	80	120	50	50	428	69
Average$_P$	199/.93	183/.87	80/.99	45/0.44	39/.71	16/.31	237/.58	22/.33
Average$_P$-global	137/.63	121/.62	80/.99	44/0.43	22/.48	14/.26	189/.48	20/.31

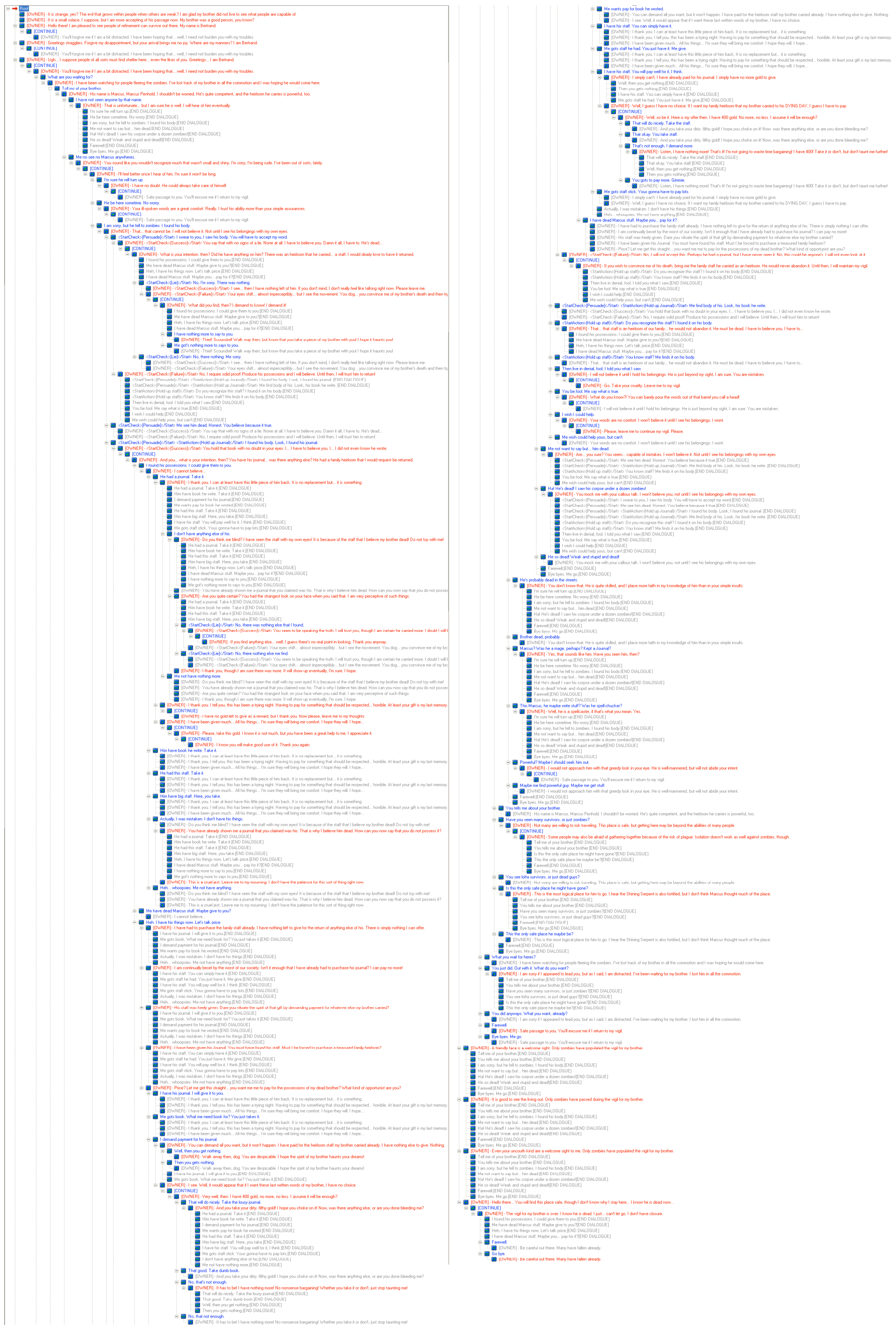


Fig. 15 The 451-line textual remark graph for the *Bertrand* conversation reduced to a single page. The 97 scripts are not shown.

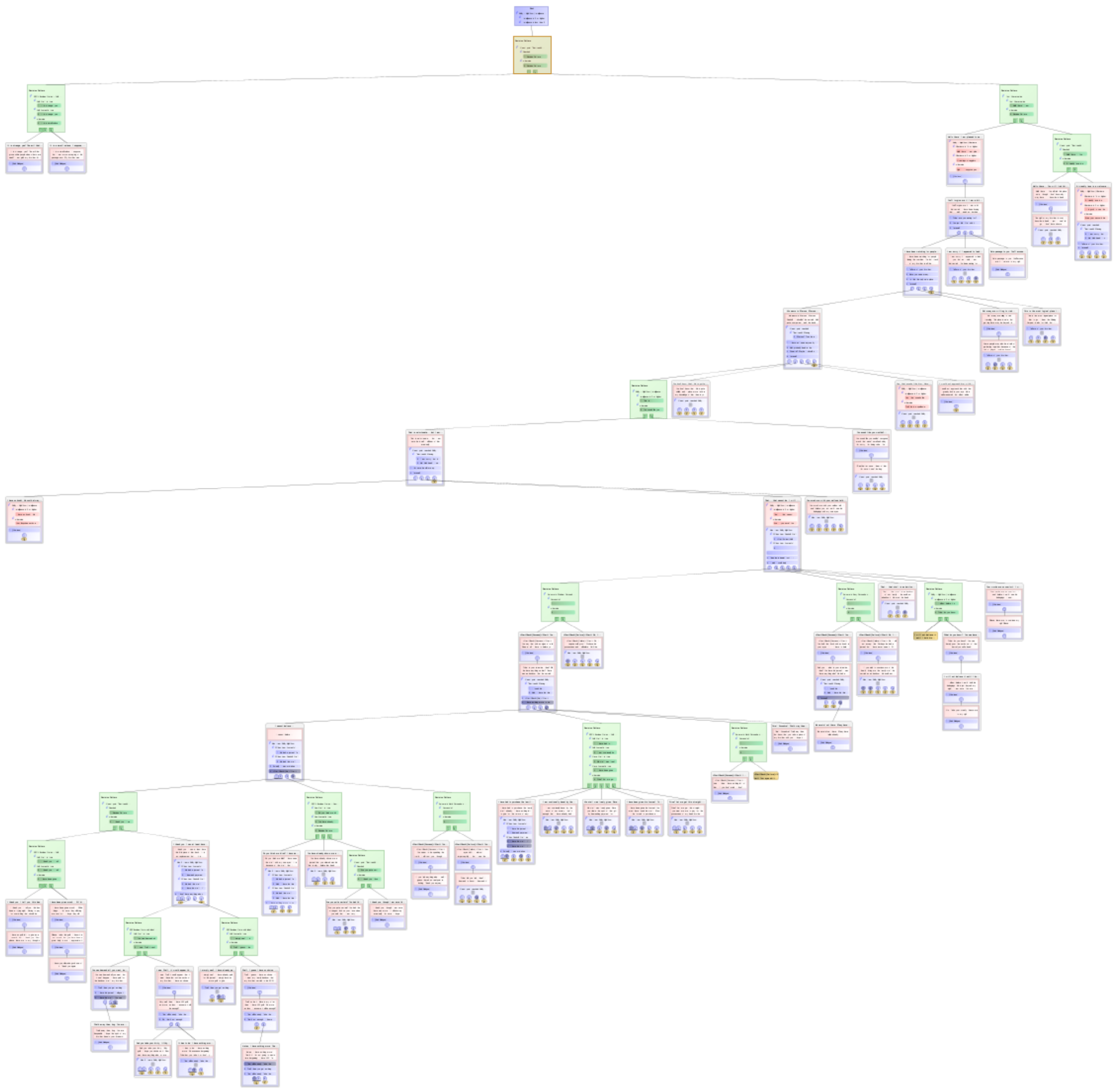


Fig. 16 The pattern graph for the *Bertrand* conversation.

The dialogue pattern model that uses a global user filter scores lower (better) than the remark conversation model in seven of the eight simplicity scores for all five conversations. For the remark score, they usually tie, since the number of remarks entered by the user is the same, except where a user remark was left unshared in the remark model (*Siri* and *Bertrand*). The *Siri* conversation has no topics with more than one exchange so the topological simplicity is the same as the structural simplicity in the pattern model. Since there are some short linear chains in *Siri*, the remark model scores slightly lower (better) than the pattern model that does not use a global user remark filter. In all other cases, the pattern model without a global user remark filter scores better than the remark model. In all cases, the pattern model scores as good as or better than the remark model if a global user remark filter is used.

Although the lower metric scores provide a good argument for pattern graphs over remark graphs, a simple human visual argument for simplicity can also be made. We shrunk the remark graphs and pattern graphs for the conversations so that each could be displayed on a single page and compared the images. The reader is not intended to read the lines in Fig. 15 or look into the boxes in Fig. 16, where the images for the largest conversation, Bertrand, are shown. Instead, these figures serve as diagrams to indicate relative complexity. Note that the remark graph has 97 scripts that are not shown, while the pattern graph contains the equivalent control information explicitly. The pattern graph approach produces a graph whose high-level structure and control are easier to visualize.

10. Summary and future research

While dialogue structure can be represented by a remark graph, a graph that contains a richer set of nodes, such as exchanges, topics and shared user menus, can provide useful abstractions that make the dialogue easier to understand and edit. Our case study has shown that pattern graphs support a useful set of components that improve the structural representation of dialogue graphs

Dialogue control semantics are difficult to write, especially for authors who are non-programmers. If dialogue control is added to a remark graph as a set of local scripts, then the control is error-prone, since lack of local scoping leads to global variables and script interactions often require awkward rules such as NWN's short-circuit semantics.

Our goal was to create a visual language that: 1) enables a non-programmer to easily specify the dialogue control semantics, 2) displays the dialogue control semantics explicitly in the dialogue graph for easier editing and maintenance, 3) automatically generates code that implements the dialogue control semantics, and 4) organizes the remarks into higher level structures than individual remarks so that the dialogue graphs are smaller and easier to create and maintain. Our case study shows that we have succeeded. Although this case study focused on dialogues for computer games, the lessons we learned are also applicable to instructional dialogues, web-based dialogues and to the general problem of creating dynamic dialogues. In fact we think it will also apply to the general problem of controlling traversals in directed graphs. One important subset of such applications is controlling game-tree search (not dialogues), in which two participants alternate moves at alternate levels of the tree. The user-agent model can be applied to game-trees in which a human player makes one set of moves and the computer makes the other and the agent-agent model works for game-trees in which the computer makes both sets of moves (self-play).

In the context of computer games, our next goal is to create a set of highly re-usable intentional dialogues that capture common situations in story-based games. Some specific examples are dialogues for quest givers, direction givers, mentors, storekeepers, guards, healers, puzzle askers and children playing. Such intentional dialogues are very domain specific, but they can be created using the structural and control patterns described in this paper. We also intend to run user studies to ensure that our metrics translate to real simplification from the user perspective.

Acknowledgement

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE). We would like to thank the members of the ScriptEase team who provided help and guidance during this research and whose broad shoulders provided the research and development that enabled the dialogue pattern research to be conducted: Jonathan Schaeffer, Maria Cutumisu, Curtis Onuszko, Kevin Waugh, Allan Schumacher, Josh Friesen, Robin Miller, Matt McNaughton, Thomas Roy, James Redford and Dominique Parker.

References

[1] Bioware Corp. http://www.bioware.com.
[2] The Elder Scrolls. http://www.elderscrolls.com.
[3] The Elder Scrolls Dialogue Editor. http://cs.elderscrolls.com/constwiki/index.php/Category:Dialogue.
[4] G. Kacmarcik, Question-Answering in Role-Playing Games. In Papers from the AAAI Workshop on Question Answering in Restricted Domains. AAAI Press (2005).
[5] G. Kacmarcik, Using Natural Language to Manage NPC Dialog. In Artificial Intelligence and Interactive Digital Entertainment (2006).
[6] M. Mateas and A. Stern, Facade: An Experiment in Building a Fully-Realized Interactive Drama. In Game Developers Conference (2003).
[7] The Sims. http://thesims.ea.com/.
[8] Owen, C.B., Biocca, F., Bohil, C. and Conley, J., Simdialog: A Visual Game Dialog Editor, Proceedings of Meaningful Play, Michigan State University, East Lansing MI, October, 2008.
[9] McRoy, S. W., and S. Ali, A Practical, Declarative Theory of Dialog, Linkoping Electronic Articles in Computer and Information Science. 4(30) Linkoping University Electronic Press. http://www.ep.liu.se/ea/cis/1999/030/
[10] Traum, D. and Larsson, S, The Information State Approach to Dialogue Management in Current and New Directions in Discourse and Dialogue, Ed. Jan van Kuppevelt and Ronnie Smith, Kluwer (2003) 325 – 346.
[11] CD Projekt. http://www.cdprojekt.com.
[12] Obsidian Entertainment. http://www.obsidianent.com.
[13] Ion Storm. http://en.wikipedia.org/wiki/Ion_Storm_Inc.
[14] Planetdeuxex.com. Tutorials. http://www.planetdeusex.com/constructor/Tutorials.htm

[15] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, MA, Addison-Wesley, 1994.

[16] 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 (2004), 88–99.

[17] 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, (2004) 129–133.

[18] 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) (2006), 88–99.

[19] D. B. West, *Introduction to Graph Theory*, Second Edition, Prentice Hall 1996, 2001.

[20] Leda. http://www.algorithmic-solutions.com.

[21] K. Mehlhorn and St. Näher, *The LEDA Platform of Combinatorial and Geometric Computing*, Cambridge University Press, 1999.

[22] M. Carbonaro, M. Cutumisu, H. Duff, S. Gillis, C. Onuczko, J. Siegel, J. Schaeffer, A. Schumacher, D. Szafron and K. Waugh, Interactive Story Authoring: A Viable Form of Creative Expression for the Classroom, Computers and Education 51 (2), September 2008, 687 - 707.