

## ABSTRACT

HARRIS, JUSTIN T. Proactive Mediation in Plan-Based Narrative Environments. (Under the direction of Assistant Professor R. Michael Young).

In interactive plan-based narrative environments, user's actions must be monitored to ensure that conditions necessary for the execution of narrative plans are not compromised. In the Zocalo system, management of user actions is performed on a reactionary basis by a process called mediation. In this thesis, an extension to this approach, proactive mediation, is described, which calculates responses to user input in an anticipatory manner. A proactive mediation module accepts as input a plan describing the actions being performed by the user (generated by a plan recognition system) and identifies portions of that plan that jeopardize the causal structure of the overall narrative. Once these portions are identified, proactive mediation generates modifications to the narrative plan structure that avoid the unwanted interaction between user and story. This extension to the original mediation algorithm provides more responses to a user's actions and generates responses that are more tailored to the user's actions.

**Proactive Mediation in Plan-Based Narrative Environments**

by

**Justin Harris**

A thesis submitted to the Graduate Faculty of  
North Carolina State University  
in partial satisfaction of the  
requirements for the Degree of  
Master of Science

**Computer Science**

Raleigh

2005

**Approved By:**

---

Dr. James Lester

---

Dr. Jon Doyle

---

Dr. R. Michael Young  
Chair of Advisory Committee

## Biography

Justin Harris was born in Chapel Hill, NC on January 14, 1981 and grew up in Asheville, NC, graduating from Asheville High School in 1999. He came back to the Triangle area to attend NC State University in the Fall of 1999, majoring in Computer Science. During his senior year, Justin enrolled in Dr. Michael Young's Game Development course, which he thoroughly enjoyed. So much so, in fact, that he decided to complete his undergraduate degree working on the Mimesis project during the summer of 2003.

Justin graduated Summa Cum Laude with BS in Computer Science in August of 2003 and immediately began his graduate work with the Liquid Narrative group. The experience and knowledge gained during this time were undoubtedly invaluable, and have played an important role in his personal and professional life. He will continue to live and work in the Triangle area for the time being, all the while pursuing his interests in computer graphics, AI, and user interaction.

## Acknowledgements

I would first like to thank my committee members Dr. Jon Doyle and Dr. James Lester, as well as my advisor, Dr. Michael Young. I would especially like to thank Dr. Young for his generous encouragement and support through my first attempts in doing research. Surely, this thesis would not be possible without it. Further, the lab as a whole has provided a wonderful foundation of ideas to build upon. Thanks go to Brian Shiver, David Burke and Alex Wood for their help in writing Crossbow, to Arnav Jhala and James Niehaus for their continued support of it, and to Yun Gyung Cheong, Byung Chull Bae and Jim Thomas for their many conversations and friendship (and ski trips!). Special thanks goes to Tommy Vernieri for all of his help with XML schemas, proper programming practices, and overall role in bringing about the birth of Zocalo.

I would also like to thank Jay Johnston and Jesse Lovelace, who were great roommates throughout the majority of my time in graduate school, and my parents, who have been so supportive in all of my various pursuits. Lastly, I owe a great deal of gratitude to Jennie Grammer, whose constant love and support has been a great source of strength during this process.

This work has been supported by National Science Foundation CAREER award 0092586 and by Microsoft Research's University Grants Program.

# Contents

<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>5</b>
2.1 Interaction in Narrative . . . . .	5
2.2 Anticipating Future State . . . . .	7
2.3 Plan Recognition . . . . .	8
<b>3 Reactive Mediation</b>	<b>9</b>
3.1 Background . . . . .	9
3.1.1 Plan Description . . . . .	10
3.1.2 Characterizing User Actions. . . . .	11
3.1.3 Responding to Exceptional Actions. . . . .	12
3.1.4 Policy Tables. . . . .	13
3.2 Limitations of Reactive Mediation . . . . .	13
<b>4 Proactive Mediation</b>	<b>15</b>
4.1 Inputs . . . . .	15
4.2 Generating the Mediated Plan and Identifying Steps . . . . .	18
4.3 Avoiding Exceptional Steps . . . . .	20
4.3.1 Proactive Intervention. . . . .	21
4.3.2 Proactive Accommodation. . . . .	24
4.3.3 Proactive Reordering. . . . .	25
4.4 Replanning . . . . .	26
<b>5 Implementation</b>	<b>27</b>
5.1 Zocalo Architecture . . . . .	27
5.2 Kyudo . . . . .	28
5.2.1 Inputs . . . . .	29
5.2.2 Algorithm . . . . .	30

5.2.3	Proactive Mediation within Zocalo . . . . .	31
<b>6</b>	<b>Example</b>	<b>33</b>
6.1	Corporate Espionage . . . . .	33
6.2	Detecting Exceptions . . . . .	36
6.3	Addressing the User's Alternate Movement . . . . .	36
6.4	Getting the Boss to His Office . . . . .	39
6.4.1	Substitution . . . . .	39
6.4.2	Aversion . . . . .	41
6.4.3	Disablement . . . . .	41
6.4.4	Reordering . . . . .	45
6.4.5	Accommodation . . . . .	45
6.5	Discussion . . . . .	45
<b>7</b>	<b>Conclusions</b>	<b>49</b>
7.1	Future Work . . . . .	50
	<b>Bibliography</b>	<b>52</b>
<b>A</b>	<b>Example Inputs</b>	<b>56</b>

# List of Figures

4.1	Step Types . . . . .	19
4.2	Substitution . . . . .	22
4.3	Aversion . . . . .	23
4.4	Disablement . . . . .	24
4.5	Accommodation . . . . .	25
4.6	Reordering . . . . .	26
5.1	Kyudo Algorithm. . . . .	31
5.2	Zocalo Architecture . . . . .	32
6.1	Example: Narrative Plan . . . . .	34
6.2	Example: Recognized Plan . . . . .	35
6.3	Example: Merged Plan . . . . .	37
6.4	Example: Accommodating the Move. . . . .	38
6.5	Example: Substitution Solution . . . . .	40
6.6	Example: Aversion Solution . . . . .	42
6.7	Example: Disablement Solution . . . . .	43
6.8	Example: Reordering Solution . . . . .	44
6.9	Example: Accommodation Solution . . . . .	46

# List of Tables

3.1	Example Policy Table . . . . .	13
6.1	Execution Times . . . . .	47
A.1	Goal State of the Problem . . . . .	56
A.2	Initial State of the Problem . . . . .	57
A.3	Domain Operators . . . . .	58
A.4	Failure Modes . . . . .	59
A.5	Directive Operators . . . . .	59

# Chapter 1

## Introduction

Recently, a number of interactive applications, including computer games, training simulations, and intelligent tutoring software involve a human user interacting with one or more agents embedded in a virtual environment. These applications often require the agents, in concert with the user, to perform coordinated sequences of novel actions structured as an unfolding story or narrative. This structuring of events can not only be put in place to provide a logical flow, but also to convey an aesthetic that can have a significant impact on the user's experience.

Approaches to these applications vary in the amount of autonomy given agents acting as characters in a story. Some allow agents to act unrestrained in an environment, allowing dramatic structure to emerge from the interactions among agents and the user [4]. Others employ a "director" agent to guide the characters along an intended narrative path [14, 20]. Still other story-based systems take a more central approach, in which a global planning system defines actions for all agents in the narrative environment [21, 34].

Planning systems can structure the narrative not only to coordinate agents' actions, but also to convey the story so that it is understandable to the user. Recent research [24, 31] suggests that the causal and temporal structures of AI plans closely resemble the mental models that people form when reasoning about plans. Similar work [8] has explored the effectiveness of using plans to describe narratives.

If such a plan-based system is used in the generation of *interactive* narratives, the role of the user comes into question. In work by Cavazza [7], the user takes the role of a

spectator, as the audience member of a digital play. She can minimally influence the virtual environment by verbally communicating with characters or manipulating objects. In other systems [21, 34, 30, 20], the user takes a more active role in the story, directing a character's actions as the narrative progresses.

If the user in such plan-based systems is allowed a significant amount of autonomy, careful attention must be paid to guarantee that she does not alter the environment in a manner harmful to the plan structure. Plans are carefully constructed so that they contain structure to guide the virtual environment towards the narrative's intended resolution. When the user is also interacting with this environment, she may be able to disrupt it to the point at which the complete plan cannot be successfully executed.

This problem is compounded when a user typically has limited knowledge of the unfolding narrative. Stories often contain elaborate settings filled with various characters and objects, and the user may be unaware of their current state or future use in the storyline. In fact, elements that are essential for certain stories, such as comedy or suspense, can *require* that the user not know about future events in order to achieve certain emotional outcomes. With this uncertainty about the progressing plot, the user may perform actions that threaten the causal structure of the narrative, either knowingly or unknowingly.

For example, a user in a crime story may willingly lock her car door in order to prevent it from being stolen, having heard of recent car-jackings on the radio. If the narrative involved a thief stealing the car at a later time, the user would have unintentionally altered the environment such that the original story will fail.<sup>1</sup> In this example, the user may believe that the future events of the narrative involve her driving her car again, or even foiling the thief and putting a halt to the rash of break-ins. Her uncertainty about future events in the story led to her performing an action that was contrary to the intended plan.

Typical solutions to this problem involve reducing or eliminating a user's autonomy in the system. However, a user's sense of engagement can be largely dependent on the amount of control that she has within the story. With more control over her actions, a user feels a stronger sense of presence [19] and is more involved with her surroundings and the general narrative. This poses an interesting problem for researchers: a balance must be made so that the user can interact as freely as possible within the environment while preserving the causal and temporal dependencies of the story. Young [32] has previously

---

<sup>1</sup>Admittedly, this is not a very good thief, but one could image a virtual world in which a thief can only enter an unlocked car.

noted this balance of story and interaction, and Aylett [3] refers to the phenomenon as the “narrative paradox”.

A previously defined process called *reactive mediation* [27] addresses this issue by pre-determining responses to destructive user behavior. In reactive mediation, a plan’s causal structure is examined and intervals within the plan are identified in which a user can perform an action that breaks some causal dependency. A list of mediation responses is generated, in which either the harmful action is substituted with a suitable alternative, or the user’s action is allowed to execute, while the system modifies the story plan such that the action is no longer harmful (see Chapter 3 for further details).

One noteworthy limitation of reactive mediation is that user’s activity is examined on a per action basis. That is, mediation responses are taken only at the point where the harmful action is performed. While preserving the validity of the plan’s causal structure, this approach fails to take into account the larger context of the user’s actions. Often, a user performs a *sequence* of actions leading to some desired result, in which one or more of those actions may be harmful to the global plan.

For instance, a science fiction narrative may involve the crew of a space ship stranded on a remote planet. Rather than venture out to explore the planet, the user may attempt to return to space by repairing the ship, which requires a number of steps that lead to the ship being operational. If later events included the crew exploring the planet and discovering a key artifact, then the act of leaving the planet prematurely would jeopardize the future storyline. In this case, reactive mediation may stop the ship from leaving at the last possible moment, after the user has spent a considerable amount of effort towards repairing the vessel. More desirable alternatives would be to guide the user away from her current plan of action or work to incorporate it more into the storyline.

In this thesis, an extension to reactive mediation, a process called *proactive mediation*, is described. Rather than examining single user actions, the proactive mediation module examines a proposed plan (provided by an external plan recognition component) that the user is performing in the context of a larger story. Having knowledge about hypothetical future actions that the user may perform allows the proactive mediation module to generate a wider variety of responses to potentially harmful user activity, as well as to shape those responses to better integrate with the overall course of the narrative.

The proactive mediation algorithm takes as input a plan describing an interactive story and a plan describing a user’s anticipated activities. The two plans are combined,

taking note of steps that the two share. Then, certain actions that the user is expected to perform are identified as harmful within the context of the story. Concepts from reactive mediation are extended to generate responses to harmful actions in the user's proposed plan. One category of response can prevent the harmful action by adding or removing plan structure so that state of the environment leaves it unexecutable. Another category allows the action to execute, but restructures the narrative so that the act is no longer dangerous.

A narrative plan often contains a complex set of causal dependencies. Given a user's plan of significant length, there could be multiple steps that may prove harmful to the story. In these cases, there are may be many possible combinations of plan modifications that eliminate the danger of these steps. A search is performed through these combinations in order to provide a new narrative that not only eliminates adverse interactions between the user and story, but also conforms to the author's narrative goals and preferences.

In the following chapters, I outline the research on proactive mediation. An overview of relevant work is given in Chapter 2, followed by background information on reactive mediation in Chapter 3. In Chapter 4, I give a formal description of proactive mediation, with a discussion of its implementation, *Kyudo*, in Chapter 5. Next, a demonstration of the algorithm is given in Chapter 6, followed by concluding remarks in Chapter 7.

## Chapter 2

# Related Work

This thesis proposes a method of proactively altering an interactive narrative in light of expected user activity. This involves coordinating system controlled characters and other aspects of the narrative world in anticipation of the user executing an expected plan. The following sections contain an overview of related research areas, including other methods of allowing human interaction in computer-based narrative, as well as the anticipation of character activity in computer-based narrative. Also, as this system relies in part on an external plan recognition component, a brief description of relevant techniques is given.

### 2.1 Interaction in Narrative

One approach to an interactive narrative was developed in Peter Weyhrauch's dissertation work on the MOE Architecture [30]. MOE allows the user to experience an interactive drama (ID), acting freely within the story environment. The user's actions, called USER MOVES, are combined with actions that the system takes, called MOE MOVES, to form a history describing the story thus far. A modified adversarial search is used to generate possible future storylines, each of which is evaluated using several features, including logical connectivity, the user's excitement, and the user's sense of freedom. These future storylines are used to select MOE MOVES that can control characters or manipulate the environment to subtly guide the plot toward the "better" future scenario.

Lamstein and Mateas [17] revive many of the ideas presented in MOE in their Search-Based Drama Manager (SBDM). SBDMs are targeted at more open-ended simulations in which a player is subtly guided so that the experience fits a favorable story arc. Like MOE, the SBDM searches possible future storylines upon the recognition of a Player Move (synonymous with USER MOVE in Weyhrauch’s work), which allows the SBDM to consider the Player Move’s impact on the entire remainder of the story. However, Player Moves only impact the narrative as they occur, with no consideration of which Player Moves are more likely when evaluating future story direction.

Management of user activity along a more structured storyline is proposed by Gordon and Iuppa [11]. Their storyline adaptation strategies define how a story, and the player’s experience of the story, reacts in response to unanticipated user action at certain choice points. Logical formalizations of commonsense psychology are used to maintain lists of the previous and future storylines, as well as previous user activity. When the user deviates from the intended storyline, these lists are examined in order to pick an appropriate adaptation strategy to modify the narrative. However, these adaptation strategies are primarily used to guide the user back onto a fixed or slightly branching story.

A user’s influence over the story’s direction is more pronounced in the Façade system [21]. Façade uses the concept of a beat to define characters’ behavior as the story progresses. The story plays out as a series of beats dynamically selected to fit a desired story arc. Each beat contains information about how characters should react to various user activity in addition to how that activity affects the likelihood of future beats.

Alternatively, a plan-based approach to interactive narrative is presented by Cavazza, Charles and Mead [7] in which a character’s behavior is represented by a Hierarchical Task Network (HTN). The user’s role in their system is that of a spectator, not of a character, and the degree of user interaction reflects that role; they can verbalize commands to the characters or manipulate objects in the virtual environment. No framework is put in place to explicitly account for user activity. Rather, agents replan their current course of action if any plan step fails, as a result of activity by the user or another character.

## 2.2 Anticipating Future State

While the above approaches to managing user activity can alter the narrative state to account for unexpected and harmful user activity, none attempt to make any guesses about future activity that may be harmful. John Laird’s work on the Quakebot [15] incorporates anticipation abilities into an agent playing Quake, a commercial video game, using the Soar cognitive architecture [16]. While this particular environment is not especially narrative oriented, it does provide evidence pertaining to the use of prediction in video games, which are often used in the production of interactive narratives [34, 7].

Anticipation of future *story* direction has been examined by Laaksolahti and Boman [14]. They present an interactive narrative framework in which autonomous character agents guide the narrative through a finite automaton over story states. A story manager examines each agent’s current goals, plans and intentions to anticipate future states of the system. If those states are “undesirable”, the story manager can perform certain actions to subtly manipulate the narrative direction, such as changing aspects of the agents’ internal model or modifying objects in the story world. However, the story manager only takes the synthetic agents’ models into account and makes no predictions of any human users’ actions.

Work by Magerko and Laird [20] does incorporate hypothesized future user behavior in the Interactive Drama Architecture (IDA) system. IDA uses a rule-based user model to predict world state changes between predefined plot points in a narrative. Their model is used to determine if a user’s expected actions are likely to satisfy the preconditions of any plot points and to adapt their execution environment accordingly to further advance the story. While their approach detects and reacts to anticipated inconsistencies in the story, the system responds to expected user actions only at the end of the simulation created by their rule-base.

In contrast, the process I define below uses an explicit plan representation to describe hypothesized user behavior. This representation not only allows for planning responses to user actions, but also identifies specific harmful actions and the conditions they require for execution. The resulting system can preemptively alter the world state and the actions the system will execute in order to prevent the user from performing any harmful actions.

## 2.3 Plan Recognition

In order to reason about the interactions between future story events and future user activity, a plan describing that activity must be submitted to the proactive mediation module. While the specifics of inferring a user's a plan lie beyond the scope of this thesis, a brief overview of relevant work in plan recognition is given (for further discussion, see the survey provided in [6]).

Plan recognition involves observing an agent's actions in order to infer their goals, intentions, or future actions, and can be partitioned into two broad categories. The first is *keyhole recognition*, which passively observes the agent whose plan is being inferred. In keyhole recognition, the agent being observed is assumed to either not be aware of the recognition process, or not attempt influence it in any way. The second category is *intended recognition*, in which the agent is aware of being observed and is understood to affect the inference. For the purpose of this thesis, it is assumed that the former method is used in providing inputs to the proactive mediation component.

The general task of plan recognition begins with a set of goals that the agent may pursue and a set of plans, called a *plan library*, which describes how the agent can achieve those goals. As the agent's actions are observed, hypotheses for plans are formed based on the action history and possible goals the agent may be trying to achieve [6]. There are many variations on this process, some involving the use of machine learning to infer goals [18] or to adapt plan libraries to a specific agent's preferences [5].

Some work in plan recognition has applied machine learning to infer a player's plan in video games [1, 10]. As stated previously, video games have been incorporated into the architecture of various interactive narrative systems, and this has implications for similar use within those systems. It should be stated, however, that proactive mediation makes no commitment to the specific techniques used to generate the user's hypothesized plan. Any algorithm that produces a plan containing the proper notations outlined in Section 4.1 should suffice.<sup>1</sup>

---

<sup>1</sup>This also applies to cases in which plan recognition is not used at all, and some other means of proposing a user's plan is employed.

## Chapter 3

# Reactive Mediation

As mentioned above, the work on proactive mediation is an extension of previous work on reactive mediation, both of which are implemented in the Zocalo system. Zocalo is a distributed, service-oriented architecture that manages the generation and execution of interactive narratives within a virtual world. Much of the conceptual work in Zocalo is carried over from the Mimesis project [34], but follows a more modular approach which allows the integration of components independent of platform or architecture. The remainder of this chapter describes the reactive mediation process as it is implemented in this framework.

### 3.1 Background

Zocalo drives the action within its story world based on the structure of a plan produced by a narrative planner. Plan execution is complicated, however, because users are relatively unconstrained with respect to the actions that they can perform in the world as the plan is being executed. The plans used by Zocalo are dependent upon user actions, both because some user actions are required for the plans to progress and because the consequences of user actions may inadvertently interfere with the world state on which the plan structure depends. As users issue commands for their characters to perform actions within the story world, these actions must be checked against the narrative plan to determine how they fit with the plan's structure. This process, called *reactive mediation*, is described

in detail in [27].

### 3.1.1 Plan Description

Mediation is designed to preserve the validity of a narrative plan as it is being executed. The plan structure used by Zocalo is similar to that of partial-order, causal link planners [22], and contains the following components: <sup>1</sup>

- A set of steps, which represent events that occur in the narrative.<sup>2</sup> Each step contains a set of preconditions and a set of effects. Preconditions must be true in order for the step to execute, while the effects are changes to the world state resulting from the step's execution.
- A set of *ordering links*, which define a partial ordering between these steps. An ordering link is written in the form  $A < B$ , and means that step  $A$  must be executed before step  $B$ .
- A set of causal links, each of which requires that a specific condition persists between steps. A causal link is written in the form  $A \rightarrow^e B$ , and means that step  $A$  establishes the condition  $e$ , which is a precondition of  $B$ .

These components are combined to form an acyclic directed graph (DAG) which represents a partial ordering of the execution of steps. These steps describe all of the events in the narrative, including those performed by the user. However, with little restriction placed on which actions the user *can* perform, it is rare that a user will execute only those events prescribed by the narrative plan. As a user issues commands to the environment, those commands are mapped to a representation identical to that of plan steps. This allows the mediation algorithm to reason about how the user's actions can interact with the narrative plan.

To help explain how this process works, it will be examined in the context of an adventure story featuring the character Indiana Jones from the classic Steven Spielberg

---

<sup>1</sup>Proposition symbols are used to discuss concepts in this thesis, while in practice, a restricted first order language is used to describe conditions in the story world.

<sup>2</sup>Here, it is appropriate to make the distinction between a step and an action. A step refers to a data structure which describes an event in the virtual world. A plan contains a set of steps, whose referent events, at the time of planning, have not yet occurred. An action refers to an event which is occurring at the present time, regardless of whether or not it is described in a plan, or whether it is performed by the system or by the user.

pictures. The particular scene involves Indiana (controlled by the user) exploring an ancient temple in the South American jungle. Deep within the temple, sitting atop a pedestal, is an idol, which is of great value to the archaeologist. The narrative plan includes the user filling a bag with sand and carefully exchanging it for the idol in order to avoid springing a trap. When the user has obtained the idol, she can safely exit the temple, ending the scene.

### 3.1.2 Characterizing User Actions.

As the user performs an action, the mediation component must characterize the act with respect to the story's requirements; actions upon which the story depends must be identified in order for the story to progress, while actions that interfere with the story's structure must be identified so that the damage that they might cause to the narrative can be avoided or minimized. By comparing each action executed by the user to the structure of the story world plan, the mediation component automatically characterizes user actions into one of three categories: constituent, consistent, and exceptional.

A *constituent* action is one that maps directly to a step in the story world plan. The action's type and temporal and causal structure all match the corresponding constraints on a step specified for user execution. In our adventure story, the user filling the bag with sand would be identified as constituent. This action was prescribed by the narrative plan, and upon completion is marked as successfully executed.

A *consistent* action is one that is not constituent and whose effects do not alter the virtual world in a way that explicitly interferes with the successful execution of the story world plan. Specifically, an action  $a$  is consistent just when it is not constituent and, for each of its effects  $e$ , there is no causal link in the story world plan that spans the point in time where  $a$  is being executed and that is labeled  $\neg e$ . An instance of a consistent action in the example adventure story would be if the user decided to walk over to one of his colleagues and ask his opinion of the situation. This action is not a part of the intended storyline, but it does not alter the environment in a harmful manner. In practice, most user actions fall into this category.

An *exceptional* action is neither constituent nor consistent, that is, at least one of the effects of the action threatens a causal link in the narrative plan. Formally, action  $a$  with effect  $\neg e$  threatens causal link  $s_1 \rightarrow^e s_2$  when  $a$  is performed after  $s_1$  and before  $s_2$ . Exceptional actions, if allowed to execute, break the causal dependencies on which a story

plan is based, making the plan impossible to execute. At the ancient temple, if the user bypassed any precautionary measures and simply took the idol off its pedestal, the act of grabbing the idol would be identified as an exception. The step in the plan that indicated that the user should replace the idol with the sand bag had causal dependencies that the idol was resting on the pedestal and that the trap was not sprung, both of which were threatened by the user's careless act.

### 3.1.3 Responding to Exceptional Actions.

When exceptional actions are initiated by the user, their execution changes the state of the story world in such a way that the story plan is no longer executable. In order to prevent this consequence, Zocalo's execution manager monitors each command sent by the user to the virtual world, characterizing it immediately as consistent, constituent or exceptional. When an exception is detected, the system determines an appropriate response before the user's command is queued for execution. The execution manager responds to each exception either by preventing the exception's threatening effect to be established or by adjusting the narrative such that the action's performance poses no threat. These outcomes are achieved by *intervention* or *accommodation*, described briefly below.

When an exceptional action is handled by intervention, an alternative action is executed in its place. This alternative action, instantiated from a set of predefined failure modes, is similar in appearance and function to the exceptional action, but has a different set of preconditions and effects. For example, when our carefree user is attempting to grab the idol from its pedestal, a nearby colleague could grab Indiana's wrist, informing the user of the trap that could be triggered. This failure mode is believable within the domain of the story, and does not have the harmful effects of the original action.

An alternate response is to accommodate the exceptional action, allowing it to execute, and restructuring the remaining plan so that no causal links are threatened. This restructuring can often be slight, such as planning for Indiana to run out of the temple, closely followed by the massive boulder unleashed by springing the trap. However, in certain cases, the revised narrative plan may be substantially different from the original, requiring significant computation on the part of the planner. If for example, the scene had involved retrieving another artifact from the temple, the entrance is now blocked by the boulder, and the user may have to perform an elaborate sequence of actions to reenter.

Action	Interval	Responses
Take(Indiana, Idol)	0:Initial - 5:Replace(Indiana, Idol, Sandbag)	GrabHand(Robert, Indiana)

Table 3.1: An example of a policy table, with a single entry describing the response to take if Indiana tries to take the idol rather than replace it with the sand bag.

### 3.1.4 Policy Tables.

The process of revising plans and finding suitable failure modes is complex, and cannot reliably execute in an acceptable amount of time if performed when the exception occurs. In order to provide satisfactory response time when an exception does occur, accommodations and interventions are generated in advance, and held in the *mediation policy table*.

After generating a narrative plan but prior to its execution, Zocalo’s mediation component examines the plan’s causal structure and identifies which user actions can cause exceptions at every point in the plan where a user may act. For every possible exception, a queue of appropriate responses (interventions and accommodations) is computed. This action/response queue pair is inserted as an entry into a mediation policy table, along with the interval in the narrative plan during when the action can be performed, as shown in Table 3.1.4. The queue of responses is sorted by a heuristic function which determines a qualitative measure of the responses’ effectiveness.

In order to respond to exceptions quickly, any accommodating plan that is generated as a response must also have an associated policy table. This is to prevent the user, through a rapid succession of exceptions, from essentially “working ahead” of the mediation component and threatening a plan before its policy table has been populated. This hierarchy of plan/policy table pairs constitutes the mediation *policy tree*, which acts as a branching storyline, expanding as the user plays through the narrative.

## 3.2 Limitations of Reactive Mediation

One significant limitation of reactive mediation is that it responds to user activity at the last possible moment. While this approach localizes the point in a story where a user’s agency may need to be restricted, intervention and accommodation at the point

of an exception can be problematic. Consider a scenario in which the user executes a long series of actions that clearly lead to an exceptional action, for instance, besieging the castle of a story's central character, capturing him and then attempting to kill him. If the system allows the user to spend the time and resources to capture the nobleman, but then intervenes repeatedly as the user swings his sword, the user will not only be frustrated with his apparent inability to hit his target but also with the failure of the system to have guided the story more effectively. The user has put in significant effort in pursuit of a particular course of action, and yet the system has done nothing to deter the user until the action sequence's very end.

In addition, it is possible for exceptions to have entries in the policy table with no generated responses. The user may perform some irreversible action which has no contingent plans or defined failure modes, or none that are applicable in the current world state. In these cases, the system's only real options are to allow the plan to fail, or report that the action is unexecutable, both of which are rather extreme reactions.

To avoid these scenarios (and provide alternatives to less desirable reactive policies), it would greatly benefit the mediation component to have the ability to preemptively modify the narrative in anticipation of possible future exceptions. In the next chapter, the process of proactive mediation is defined, which attempts to do just that.

## Chapter 4

# Proactive Mediation

The problems with reactive mediation are addressed by proactive mediation, which preemptively restructures the narrative plan to better account for anticipated user activity. Rather than monitoring the user's activity only as it occurs, a proactive mediator also examines the user's anticipated plan of action provided by an external plan recognition component (e.g., [1, 10]). Steps in the user's anticipated plan are characterized as constituent, consistent, or exceptional, just as individual actions are categorized under reactive mediation. Proactive mediation extends the notions of intervention and accommodation to avoid threats from exceptional steps that have not yet occurred. While the basic objectives of reactive and proactive mediation are the same, a proactive mediator's knowledge of expected future steps is utilized to allow a wider range of responses to user activity.

### 4.1 Inputs

As described in the previous chapter, Zocalo uses a plan representation to describe story events in a virtual environment. The proactive mediation component uses this same plan representation to describe likely future event sequences that the user may perform. This plan is supplied by a plan recognition component which, upon recognizing a user's plan, submits it to the proactive mediation component. A plan is defined formally below.

**Definition 1 (Plan):**

A plan is a tuple  $\langle S, O, L \rangle$  where  $S$  is the set of steps,  $O$  is the set of ordering links between steps in  $S$ , and  $L$  is the set of causal links between steps in  $S$ . The proactive mediator takes as input a *narrative plan*, a *recognized plan*, a *narrative operator library*, a *directive operator library*, and a set of *failure mode definitions* defined formally below.

**Definition 2 (Narrative plan):**

A narrative plan is a plan  $N : \langle S_N, B_N, O_N, L_N \rangle$  generated by the Zocalo system which describes all of the steps to be performed by the characters in a story, including those of the user. We say that a step  $s$  is a narrative step just when  $s \in S_N$ .

**Definition 3 (Recognized plan):**

A recognized plan is a plan  $R : \langle S_R, B_R, O_R, L_R \rangle$  that is proposed by a plan recognition system. This plan hypothesizes the sequence of steps that the user intends to perform, and that the user expects to occur. A step  $s$  is a recognized step just when  $s \in S_R$ .

Each step in either the narrative plan or the recognized plan can be a *system step* or a *user step*, and is identified by a unique ID. A system step is any step executed by system resources, characters, etc. A user step is initiated by the user and performed by the user's character.<sup>1</sup> The set of system steps is denoted  $S_{SYS}$ , where  $S_{SYS} \subseteq (S_N \cup S_R)$ . The set of user steps is denoted  $S_{USER}$ , where  $S_{USER} \subseteq (S_N \cup S_R)$  and  $S_{SYS} \cap S_{USER} = \emptyset$ .

Several assumptions are made about the recognized plan. First, it is assumed that the recognized plan is executable in the context of the narrative plan. That is, there are no conflicts between narrative steps and recognized causal links. In the case that the plan recognition component recognizes a plan that can fail, the input to the proactive mediation component is the portion of that plan up to the point of failure. This input verification processing can be performed by the plan recognition component or by a separate preprocessing component.

Also, it is assumed that steps in the recognized plan are either steps that the user intends to perform or steps that will occur in the narrative. That is, the recognized plan will not include steps that the user anticipates (but does not control) that are not in the

---

<sup>1</sup>In more general terms, this set of steps can more accurately be described as *environment initiated steps*. That is, any step that is not a system step, and therefore originates from the execution environment. In the Zocalo system, all user activity is understood to fall into this category.

narrative plan. Stated formally,  $(S_R - S_N) \cap S_{SYS} = \emptyset$ . This is not to say that the user is not allowed to make incorrect guesses, but any such guesses (and any plan structure relying on those guesses) are removed from the input in the verification process described above.

It should be emphasized that the recognized plan represents speculative user activity. Predictions about the future are always prone to inaccuracies, due not only to the plan recognition component, but also to human indecisiveness or distraction. Because of this limited reliability, steps in only the recognized plan are used purely to aid refinements to the narrative plan.

**Definition 4 (Narrative Operator library):**

A narrative operator library is a collection of operators characterizing the actions available for the given story world domain, instantiated as steps. An operator is a tuple  $\langle P, E \rangle$  where  $P$  is a set of preconditions that must hold true before the step is executed, and  $E$  is a set of effects that are true after the step is executed.

**Definition 5 (Directive Operator Library):**

A directive operator library is also a collection of operators, with the same structure as narrative operators. The primary difference is in their use, which places additional restrictions on their inclusion in the plan and execution characteristics. Directive operators are used to guide the user by manipulating the world state such that the user cannot perform harmful actions. They are not included in the narrative plan when it is initially constructed, but added in by the proactive mediation component (see the sections on Substitution and Aversion responses below) and are all assumed to be system actions.

Due to their intended usage, steps instantiated from directive operators are assumed to execute “instantaneously”. This is no different than assumptions made in traditional STRIPS style planning [25], however this is important to highlight in contrast to the narrative operators, in which this assumption is relaxed. This restriction is necessary due to the ability of the user to act relatively unconstrained in the environment. If the world state must be changed to prevent the user from performing some anticipated harmful action, instantaneous execution avoids a possible race condition between the system and the user. An example of a directive operator would be *Lock*, which would programmatically lock a door in the environment without requiring that a character perform the act. Addi-

tional work in employing principles of temporal planning [2] could allow this constraint to be dropped.

In many environments, liberal use of directive operators could provide for a rather confusing experience. If a user were to open a book, curious of its contents, only to watch it close itself immediately, she will most likely be very aware of the mediation process and frustrated at the system's harsh reaction. However, this potential problem should be addressed by the author of the interactive narrative, if indeed the author deems it undesirable. Preconditions concerning the user's knowledge of the book's state, or whether the user can observe the book as it closes will ensure that directive operators are only used when "appropriate".

**Definition 6 (Failure Mode Definition):**

As stated in the description of reactive mediation, a failure mode is an operator that describes an alternative to a narrative operator. A failure mode definition is a tuple  $\langle O, F, P \rangle$  where  $O$  is a narrative operator,  $F$  is an alternate operator that can be substituted for  $O$ , and  $P$  is a set of parameter mappings from  $O$  to  $F$ . The set of failure mode definitions is denoted  $D$ . It should be noted that there are no requirements regarding common preconditions or effects among  $O$  and  $F$ . Rather, it is the responsibility of the author or knowledge engineer to define failure modes appropriate for the narrative domain.

## 4.2 Generating the Mediated Plan and Identifying Steps

The first step in the proactive mediation process is the creation of a working plan that encapsulates the actions of both input plans. This new plan, called the *mediated plan*, is formed by merging the narrative and recognized plans. The assumption is made that a step in the narrative plan and a step in the recognized plan whose IDs are identical refer to the same event, and that the resulting mediated plan is sound. The mediated plan is thus a tuple  $M = \langle S_M, B_M, O_M, L_M \rangle$  where  $S_M = S_R \cup S_N$ ,  $B_M = B_R \cup B_N$ ,  $O_M = O_R \cup O_N$  and  $L_M = L_R \cup L_N$ .

Once the mediated plan is created, the steps in the recognized plan are then categorized as *inclusive* or *exclusive*. Inclusive steps occur in both plans (i.e.,  $N$  and  $R$ ), while exclusive steps only occur in the recognized plan. Inclusive steps may be either user

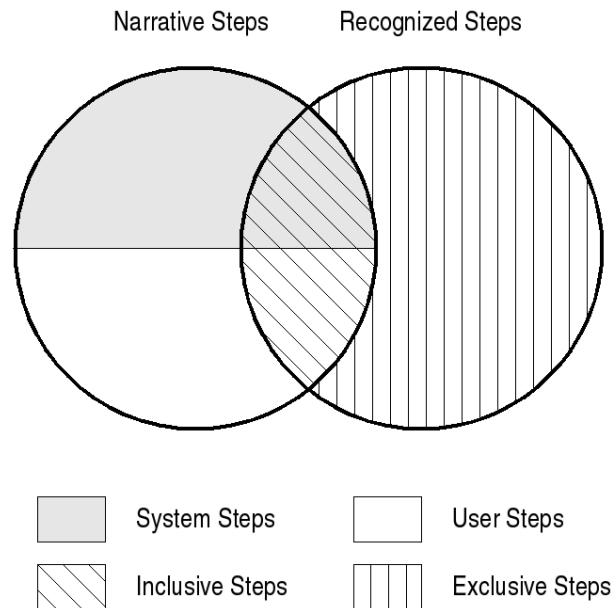


Figure 4.1: Step Types: The circle on the left represents steps in the narrative plan, while the circle on the right represents steps in the recognized plan. The gray area represents system performed steps, while the white area represents user performed steps. The area with diagonal stripes represents inclusive steps, while the area with vertical stripes represents exclusive steps.

steps or system steps, while exclusive steps are assumed to be only performed by the user. That is, it is assumed that the user will not plan for system steps to occur which are not actually part of the narrative plan.

A step  $s$  is an inclusive step just when  $s \in S_R \cap S_N$ . The set of inclusive steps is denoted  $S_{IN}$ . A step  $s$  is an exclusive step just when  $s \in (S_R - S_N)$ . The set of exclusive steps is denoted  $S_{EXL}$ .

User steps can be constituent, consistent, or exceptional, just as user actions. The definitions of these steps are similar to their action counterparts, with the understanding that the steps refer to hypothetical future events. All inclusive steps that are performed by the user are identified as constituent, and all exclusive steps are identified as either consistent or exceptional. These definitions are formally defined below:

**Definition 7 (Constituent Step):**

A step  $s$  is constituent just when  $s \in S_{IN} \cap S_{USER}$ .

**Definition 8 (Exceptional Step):**

A step  $s$  with effect  $\neg e$  is exceptional just when

- $s \in S_{USER}$  and
- There exists a causal link  $s_1 \rightarrow^e s_2 \in L_N$  such that  $s$  may possibly occur after  $s_1$  and before  $s_2$  given the constraints in  $O_M$

The set of exceptional steps is denoted  $S_{EXP}$ .

**Definition 9 (Consistent Step):** A step  $s$  is consistent just when  $s \in S_{EXL} - S_{EXP}$ .

One distinction between action and step definitions is in that of an exceptional step, which can be described as a potential exceptional action given the ordering constraints of the mediated plan. Because the mediated plan is partially ordered, it may be possible that some linear ordering of steps avoids the exception. In other words, the exceptional step may not be explicitly ordered after the beginning or before the end of the threatened causal link. However, since there is a possibility that the step could threaten the causal link in question, the step is treated as exceptional.

### 4.3 Avoiding Exceptional Steps

Once the steps in the mediated plan have been characterized, the proactive mediator then determines how to respond to each exceptional step. We say that an exceptional step is avoided when the mediator alters the narrative plan in a manner that deals with the harmful effects of the exceptional step. For each exceptional step  $s_x \in S_M$  with effect  $\neg e$  that threatens some causal link  $s_1 \rightarrow^e s_2 \in L_M$ ,  $s_x$  can be avoided by using:

- *Proactive Intervention:* Stop the user from performing step  $s_x$  in the mediated plan.
- *Proactive Accommodation:* Eliminate the need for the causal link  $s_1 \rightarrow^e s_2$  in the mediated plan.
- *Proactive Reordering:* Enforce orderings such that  $s_x$  cannot occur between  $s_1$  and  $s_2$ .

### 4.3.1 Proactive Intervention.

The purpose of intervention is to prevent the exception's threatening effect from being established. Reactive intervention achieves this by replacing the execution of the exceptional action with the execution of one of its failure modes that does not have the threatening condition as an effect, which is also possible under proactive intervention. However, since an exception considered by the proactive mediator has not yet occurred, additional action can be taken by the system to make one or more preconditions of the exception false, thus making the action itself un-executable.

Proactive intervention prevents the user from performing some exceptional step  $s_x$  in question. This can be done by stopping the execution of  $s_x$  itself, or any step that *contributes* to  $s_x$ .

**Definition 10 (Contributory):**

A step  $s_1$  contributes to  $s_x$  just when there exists  $s_1 \rightarrow^e s_x \in L_M$  or some other step  $s_2$  contributes to  $s_x$  and there exists  $s_1 \rightarrow^e s_2 \in L_M$ . Any step that contributes to an exception is said to be *contributory*.

When a contributory or exceptional step  $s$  is stopped via intervention, it is no longer expected to be executed, and is removed from the mediated plan, along with any causal links and ordering links associated with  $s$ . In addition, all steps that  $s$  contributes to are removed. The execution of  $s$  in the mediated plan can be avoided via intervention by one of the following measures:

**Substitution:** Substitution prevents the exceptional step from establishing the threatening effect by replacing a contributory step  $s$  with one of its failure modes  $s^*$ . The failure mode must be selected so that at least one of the conditions established by  $s$  and used in a contributory causal link is not asserted by  $s^*$ . Substitution can be used to prevent  $s$  by replacing  $s$  with  $s^*$  if all of the following conditions hold:

1.  $s \in S_{EXL}$
2.  $s$  is instantiated from operator  $A$  with effects  $E_A$
3.  $s^*$  is instantiated from operator  $A^*$  with effects  $E_{A^*}$
4. There exists a tuple  $\langle A, A^*, P \rangle \in D$

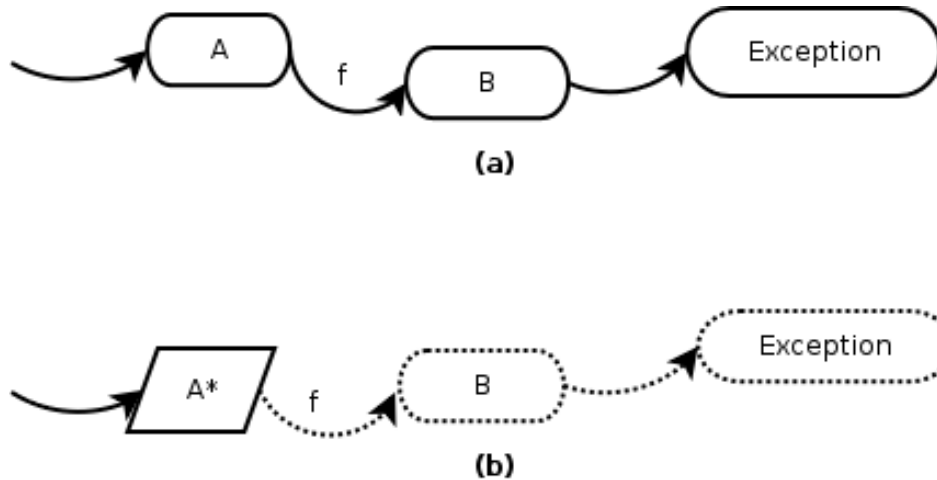


Figure 4.2: Substitution: A portion of a mediated plan with user steps  $A$ ,  $B$ , and  $Exception$  is shown in Figure 4.2(a). In order to stop  $Exception$  from being executed via substitution, step  $A$  is replaced with its failure mode  $A^*$ , as shown in Figure 4.2(b).  $A^*$  does not establish the effect  $f$ , a precondition of the contributory step  $B$ , and the causal chain is broken. Removed plan structure is shown with dashed lines.

5. For all causal links  $s_1 \rightarrow^e s_2 \in L_N$ ,  $s^*$  does not threaten  $s_1 \rightarrow^e s_2$
6. There exists  $s \rightarrow^e s_c \in L_R$ , where  $s_c$  contributes to  $s_x$  or there exists  $s \rightarrow^e s_x \in L_R$ , and  $e \notin E_{A^*}$

If the substituted failure mode  $s^*$  has any preconditions which are not in the original step  $s$ , those preconditions are considered open, requiring additional plan construction. Due to the possible race condition resulting from replanning with narrative operators, only directive operators are used to instantiate steps in this replanning process.

**Aversion:** The use of aversion prevents the execution of  $s$  by making one or more preconditions of  $s$  false prior to its execution. This is achieved by inserting a directive step  $s_i$ , called an inversion step, into the plan. Step  $s_i$  has an effect  $\neg f$ , where  $f$  is a precondition of  $s$ . Additional ordering constraints are added such that  $s_i$  must come after all steps which establish  $f$  and do not come after  $s$ , including the original source of the causal link. If the resulting plan's ordering constraints are inconsistent, aversion using  $s_i$  cannot be performed. If  $s_i$  has any preconditions, those preconditions are considered open, and the plan is refined using the directive operator library in the same manner as substitution. Aversion can be used to prevent  $s$  by inserting the inversion step  $s_i$  if all of the following conditions hold:

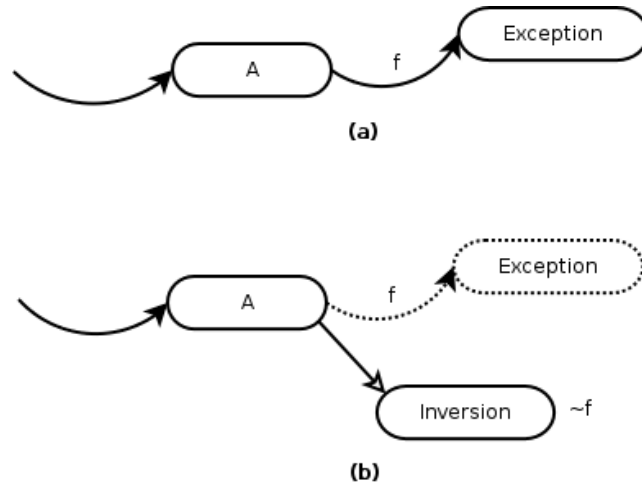


Figure 4.3: Aversion: A portion of a mediated plan is shown in Figure 4.3(a), with step  $A$  contributing to  $Exception$ . In Figure 4.3(b), aversion is used adding an  $Inversion$  step to the plan, which establishes  $\neg f$ , removing the condition required to execute  $Exception$ . The ordering link  $A < Inversion$  (shown as a white arrow) ensures that the  $Inversion$  indeed negates  $f$ . Removed plan structure is shown with dashed lines.

- There exists  $s_A \rightarrow^f s \in L_M$
- $s_A \in S_{IN} \cap S_{SYS}$
- $s_i$  has an effect  $\neg f$

In using aversion, it is important to reiterate the assumption made about the reliability of exclusive steps being executed. If any of the steps establishing  $f$  considered above are exclusive (including the source of the causal link), inverting  $\neg f$  is not performed. This is because it is undesirable to order steps in the narrative plan around recognized steps, in the event that the recognized steps are not executed.

**Disablement:** Disablement removes  $s$  from the mediated plan if  $s \in S_{IN} \cap S_{SYS}$ . This prevents  $s$  from establishing causal links which contribute to an exceptional step. Once  $s$  has been removed (along with all causal links and ordering links relating to  $s$ ), conditions in the narrative plan that were satisfied by  $s$  are no longer satisfied, and some replanning (using the narrative operator library) will be required to reestablish those conditions.

Simply removing the enabling step  $s$  is not sufficient in preventing one or more of its effects from being established. Portions of the narrative plan must be regenerated, and there is no guarantee that the undesirable condition is not reestablished in the replanning

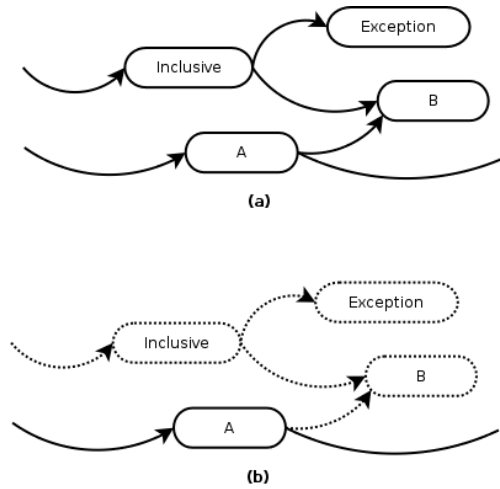


Figure 4.4: Disablement: A portion of a mediated plan, with story steps  $A$  and  $B$ , an *Inclusive* step, and an *Exception* is shown in Figure 4.4(a). The *Exception* is disabled by removing the *Inclusive* step, which contributes to *Exception*, as shown in Figure 4.4(b). Step  $B$  is also removed from the plan as part of the replanning process because it causally relies on *Inclusive*. Removed plan structure is shown with dashed lines.

process. To alleviate this, a *preservation link* is added to the plan, which ensures that the undesired effect of  $s$  does not hold. The syntax of the preservation link, and its treatment by the planning algorithm, are identical to that of a causal link. The distinction is made due to the difference in meaning and purpose for inclusion in the plan. Both guarantee that a condition holds over a given interval, but a preservation link does not imply causality.

### 4.3.2 Proactive Accommodation.

For some exceptional step  $s_x \in S_M$  with effect  $-e$  that threatens some causal link  $s_1 \rightarrow^e s_2 \in L_M$ , proactive accommodation allows the user to perform the exceptional step  $s_x$ . In response, the system replans the narrative to reestablish any causal links threatened by the effects of  $s_x$ . In this regard, there is no difference between proactive accommodation and reactive accommodation, and the basic mechanism does not differ between the two. The fundamental difference between proactive accommodation and reactive accommodation is that proactive accommodation can alter the narrative steps which occur prior to  $s_1$ . By modifying steps prior to  $s_1$ , proactive accommodation can eliminate extraneous steps from being executed to establish the original causal link, resulting in a narrative plan that is potentially more coherent. Further, plans generated with proactive accommodation can use

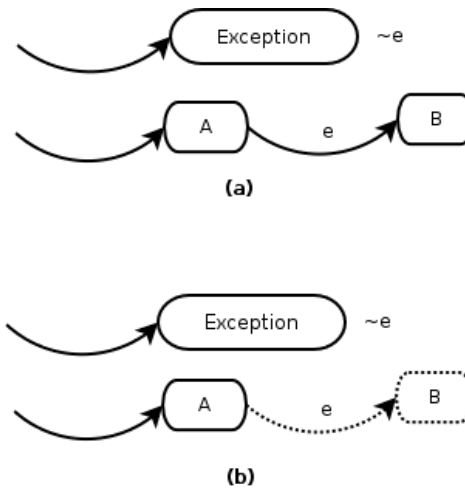


Figure 4.5: Accommodation: A portion of a mediated plan containing narrative steps  $A$  and  $B$ , and an *Exception*, is shown in Figure 4.5(a). The *Exception* is accommodated in Figure 4.5(b) by eliminating the causal link  $A \rightarrow^e B$ , along with step  $B$ , as part of the replanning process. Removed plan structure is shown with dashed lines.

effects of exclusive steps in the recognized plan to satisfy any open preconditions, potentially giving the user a stronger sense of participation in the story.

### 4.3.3 Proactive Reordering.

Proactive Reordering introduces additional orderings to the narrative plan that make it temporally impossible for  $s_x$  to be executed between  $s_1$  and  $s_2$ . Conceptually, there are two ways to enforce this: add the ordering  $s_x < s_1$ , or add the ordering  $s_2 < s_x$ . Both of these options effectively reorder the steps to prevent  $s_x$  from being executed between  $s_1$  and  $s_2$ , but neither option is particularly viable.

If the ordering  $s_x < s_1$  is introduced into the mediated plan, Zocalo's execution manager would wait for the user to perform  $s_x$  before executing  $s_1$ . This rigid requirement can easily stall the entire narrative if the plan recognition component proposed an inaccurate plan, or if the user simply changed her mind about performing  $s_x$ .

Enforcing the ordering  $s_2 < s_x$  can be problematic as well, for the simple reason that  $s_x$  is to be performed by the user. Stopping the user from executing  $s_x$  if attempted before  $s_2$  is already accomplished by reactive intervention.

There is a case, however, in which system steps can be reordered such that  $s_x$  can

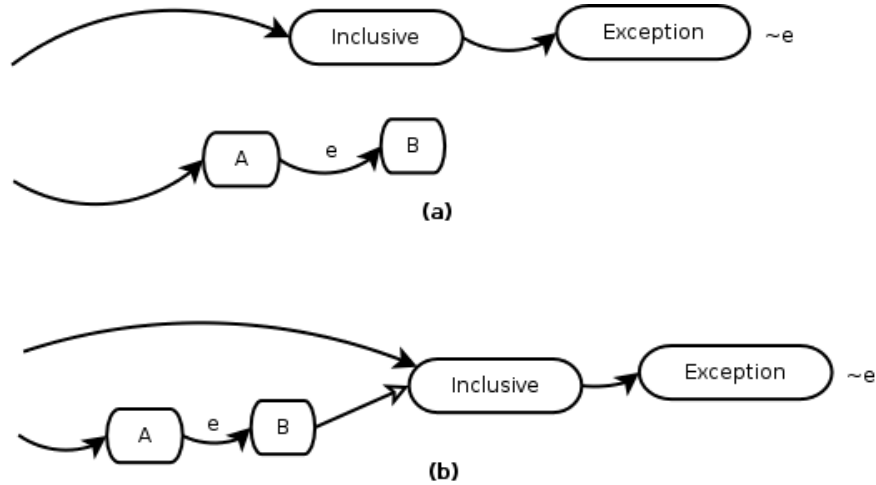


Figure 4.6: Reordering: A portion of a mediated plan, with narrative steps  $A$  and  $B$ , an *Inclusive* step, and an *Exception* is shown in Figure 4.6(a). The ordering link  $B < Inclusive$  (shown as a white arrow) is inserted into the plan so that the *Exception* cannot be executed between  $A$  and  $B$ , as shown in Figure 4.6(b).

only be executed after  $s_2$ . If some  $s_c$  in  $S_{IN} \cap S_{SYS}$  is ordered before  $s_x$ , adding the ordering  $s_2 < s_c$ , implicitly ensures that  $s_x$  cannot occur before  $s_2$  (assuming that the ordering is consistent with the mediated plan).

## 4.4 Replanning

A number of mediation strategies described above involve removing elements of the mediated plan and filling in the resulting gaps with alternative plan structure. The replanning process used to fill in the missing plan structure is similar to the plan generation process used, with one notable difference: no ordering link  $s_n < s_e$  can be added to the plan, where  $s_n$  is a narrative step and  $s_e$  is an exclusive user performed step. This restriction is in place because the system has no direct control over when  $s_e$  will be performed, since its execution is left up to the user. As a result, the execution of  $s_e$  cannot be guaranteed to follow that ordering. Ordering links and causal links are, however, allowed from exclusive steps to narrative steps, which can act to further involve the user in the narrative by effectively making exclusive actions inclusive.

## Chapter 5

# Implementation

### 5.1 Zocalo Architecture

The Zocalo system architecture is a distributed, service-oriented approach to the generation and execution of interactive narratives within virtual environments. Components of the system communicate via XML across the internet to reason about high-level narrative structure.

In its current implementation, Zocalo is comprised of three components:

- *Fletcher*: The web service providing narrative planning functionality using Crossbow, a narrative planning system based on the Longbow planner. [33]
- *Execution Manager*: The central component that serves as the gateway between the Zocalo web services and the execution environment. The Execution Manager contains logic for translating declarative descriptions of the narrative produced by Fletcher into execution messages for the environment and schedules when to send these messages. It also receives messages from the environment concerning user activity and enforces mediation policies provided by the two mediation services.
- *CrossWind*: The web service providing reactive mediation functionality [27]. CrossWind takes as input an initial narrative plan, an operator library (including failure modes), and an *action specifiers* document, which identifies environment initiated ac-

tions to be considered for mediation. Upon initiation, CrossWind can then begin the mediation session.

When a mediation session begins, CrossWind constructs the initial policy table, populating it with interventions only. Once the Execution Manager receives this policy table, execution of the narrative can begin. At this point CrossWind supplements the initial table with accommodations, calling on Fletcher to fill in missing plan data to restructure the narrative. When all accommodations have been generated, the Execution Manager updates its working policy table to the revised table that includes those accommodations. CrossWind then generates policy tables for accommodation plans in a similar fashion, producing a policy tree. In its current implementation, CrossWind expands this policy tree using an iterative depth-first search, pruning portions of the tree when they are no longer needed (when the user has passed the point in the narrative that contingent plans addressed).

The final key component (though not a part of Zocalo) is an execution environment, which actualizes execution messages from the Execution Manager and reports on user activity, as well as the success or failure of requested actions. It acts as the “front-end” to Zocalo and handles interactivity with the user such as rendering graphics and processing keyboard/mouse events. Execution environments can range from text-based applications to web based environments (Java applets, Flash pages, DHTML, etc.) to handheld platforms (Palm, PocketPC) to commercial video game engines (UT2004, Half Life 2, Neverwinter Nights) to full Virtual Reality simulations.

## 5.2 Kyudo

The implementation of the proactive mediation algorithm is named Kyudo, and is written in C# [9]. Due to its coupling with plan recognition software, it can be referenced as a library or as a separate web service. In either configuration, the plan recognition software receives notifications of user action from the Execution Manager as they occur, and uses this history to propose a recognized plan as input to Kyudo.

### 5.2.1 Inputs

The specific inputs to Kyudo can be divided into two categories: *session* inputs and *event* inputs. Session inputs are presented when an interactive session begins and remain constant throughout its duration, while event inputs are called periodically when a user's plan has been recognized.

The session inputs are:

- **Domain Specification:** The set of narrative operators, described in XML, that are used to read in the narrative plan as well as to replan the narrative when responding with accommodation or disablement. The domain specification also defines failure modes for these operators.
- **Problem Specification:** The initial state of the narrative world and the desired goal state, also described in XML.
- **User Actions Specification:** An XML document describing sets of operators that constitute user performed steps. Kyudo uses this document to identify user steps in the narrative and recognized plan inputs.
- **Directive Operators:** The set of directive operators that are used to instantiate inversion steps and to replan the narrative when responding with substitution or aversion. The file format is identical to that of the domain specification. However, since all directives are system steps, there is no need for failure modes and hence none are included.
- **Planning Heuristic:** A string indicating the heuristic to use for replanning. In its current implementation, Kyudo also uses this heuristic as the basis for the mediation heuristic, so that mediation responses that coincide with the original narrative plan are favored. While this practice may be a sensible rule of thumb, it is certainly not a requirement, and future iterations will also allow the mediation heuristic to be specified independently.
- **Replanning Search Size:** An integer representing the maximum number of nodes to search in the plan space when replanning. Planning in general is computationally complex [29], and limiting the search space keeps the mediation algorithm from becoming intractable.

The event inputs are:

- **Narrative Plan:** The steps, ordering links and causal links that constitute the narrative plan, described in XML. This represents the desired narrative, not necessarily the one in progress. It can be the result of a reactive accommodation, a previous Kyudo output, or a different plan altogether.<sup>1</sup>
- **User Plan:** The steps, ordering links and causal links that constitute the proposed plan that the user is pursuing, also described in XML.

### 5.2.2 Algorithm

A single mediated plan can potentially contain multiple exceptions, which must all be avoided before the plan can become the active storyline and begin execution. The mediation algorithm used to avoid all of the exceptions is similar in many respects to our planning algorithm, which can be characterized as a *refinement search* through a plan space graph [13]. A node in the graph represents a partial plan, and a node’s children are refinements of a specific flaw in the parent. Similarly, the mediation algorithm is a refinement search through a mediation space graph, where a node represents a plan and its corresponding policy table, and a node’s children are responses to a specific exception. Search through this mediation space is guided by an author-defined heuristic, which is used to qualitatively evaluate each plan/policy table pair. This heuristic could be derived from the same planning heuristic used to generate the original story plan, so that proactive responses that coincide with the author’s intended story are favored.

While expanding the mediation space tree, choosing an exception to avoid is not completely arbitrary. Avoiding an earlier exception can (in the case of intervention) implicitly avoid any exceptions that are causally dependent on the former, so only candidate exceptions that have no contributory exceptions are chosen.

---

<sup>1</sup>One example of a different plan might be a modification of a previous call to Kyudo. This modification might be the removal of some plan structure (such as directive steps) that are no longer needed due to the user no longer pursuing a previously recognized plan.

- Step 1: Merge narrative and recognized plans into the mediated plan
- Step 2: Set mediated plan as root of the mediation space
- Step 3: Best plan = mediated plan
- Step 4: If best plan has no exceptions, return best plan
- Step 5: Pick an exception from the best plan
- Step 6: Add to the mediation space all fixes for the exception
- Step 7: If the mediation space fringe is the empty set, return FAIL
- Step 8: Best plan = lowest ranked plan from the mediation space fringe
- Step 9: Go to Step 4

Figure 5.1: Kyudo Algorithm.

### 5.2.3 Proactive Mediation within Zocalo

Kyudo's strength lies in addressing a user's sequence of actions within the context of a narrative plan. This functionality is meant to augment, not replace, existing reactive mediation in the Zocalo framework. Both methods have their benefits, as a single action performed by the user may be an exception, whether part of a larger plan or not.

The Zocalo architecture with proactive mediation is depicted in Figure 5.2. The Execution Manager relays all user activity, along with the currently executing plan, to the plan recognition component. When the plan recognition component formulates a plan hypothesis, it proposes this plan to Kyudo and submits the output as the new narrative plan. The plan portion of this mediation node is adopted as the current plan by the Execution Manager, and the entire node is submitted to CrossWind. If the node's policy table is empty (no substitutions were made in the proactive mediation process), then the process of building the policy table is the same as that at the beginning of the Zocalo session. Otherwise, the policy table's existing entries are preserved, and additional entries are added to preserve the new plan's causal links as described in the Background chapter. When CrossWind has constructed the initial policy table, execution of the revised narrative can begin.

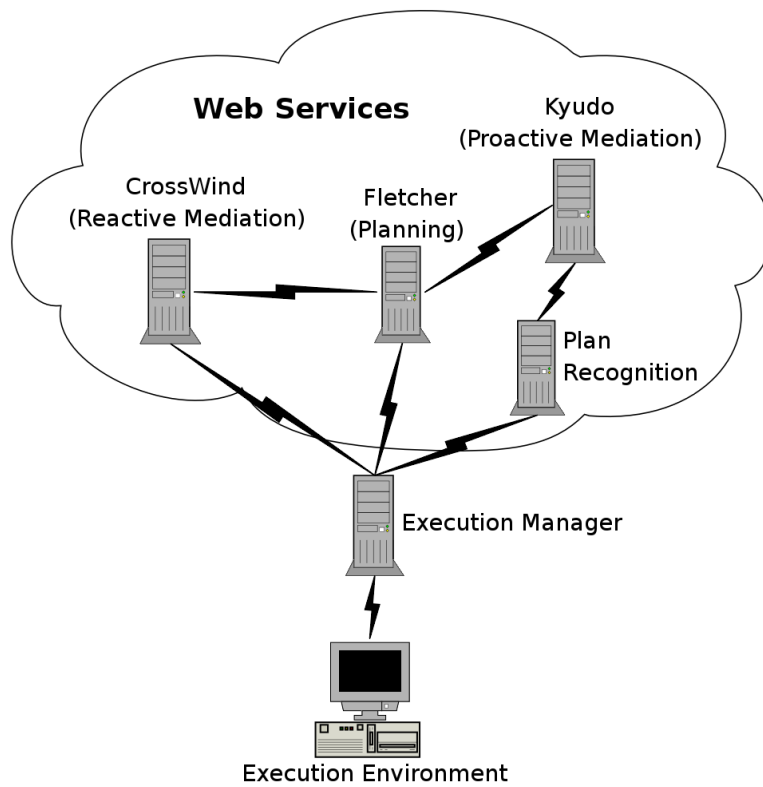


Figure 5.2: The Zocalo Service-Oriented Architecture with Proactive Mediation

## Chapter 6

# Example

### 6.1 Corporate Espionage

The following scenario is an example of Kyudo's execution. The original narrative plan describes the events in a single chapter of a larger story, in which a secret agent played by the user is attempting to acquire secret documents by posing as an employee of a corporation. The chapter ends with the user being caught in her boss's office looking for the documents. The scenario begins with the user being given a master key to the building by a collaborator on the inside (*Spy*), and then walking to her boss's office and using the master key to enter. While this is in progress, the boss arrives in his car, takes the elevator to the seventh floor, and then enters his office. This narrative is depicted in Figure 6.1.

The clever user, knowing that the boss rides the elevator every morning, decides to sabotage the elevator so that she will not be caught. The recognized plan representing her intended course of action is shown in Figure 6.2. She plans on taking the master key, walking to the power room door and entering the power room. The power room contains the controller for the elevator that the boss will be riding, so the user can break that controller to keep the boss from riding it to the seventh floor. This line of action is possible because the user will be given the master key, which she can use to open the power room door. After breaking the elevator's controller, the user can then walk to the boss's office and search for the secret documents without fear of being caught.

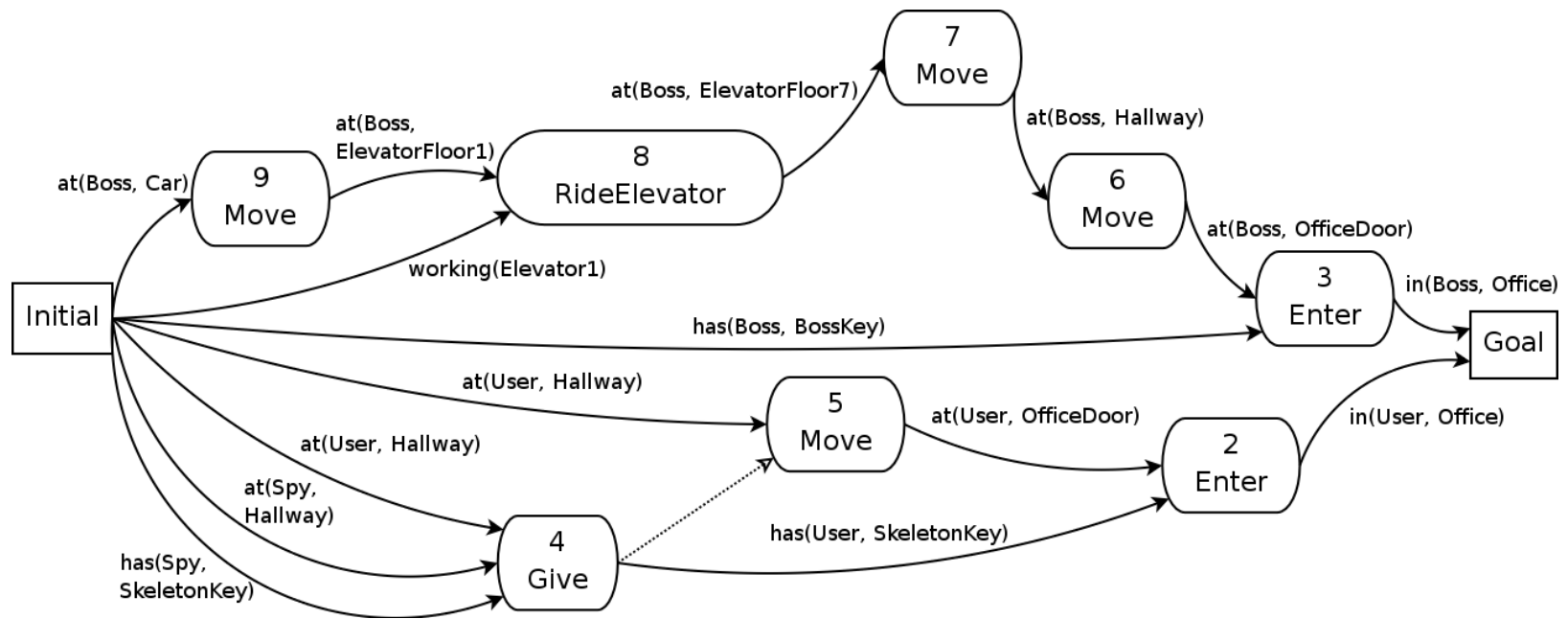


Figure 6.1: The original narrative plan, in which the user is caught in the boss's office looking for secret documents.

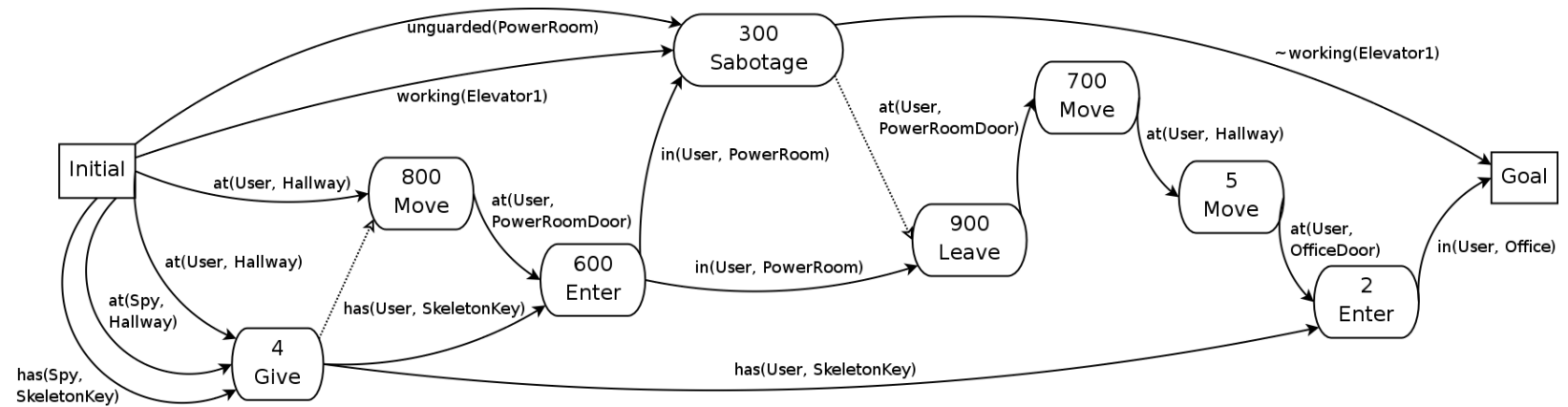


Figure 6.2: The user's recognized plan, in which the user sabotages the elevator that the boss rides every morning.

## 6.2 Detecting Exceptions

Combining the narrative and recognized plans, Kyudo constructs the merged plan depicted in Figure 6.3. Note that steps 2 (*Enter*), 4 (*Give*) and 5 (*Move*) are inclusive; they appear in both the narrative and recognized plans as both involve the spy giving the master key to the user, the user moving to the office door from the hallway and the user entering the office using the master key. Also, steps solely from the recognized plan, 300 (*Sabotage*), 600 (*Enter*), 700 (*Move*), 800 (*Move*) and 900 (*Leave*) are marked as exclusive. These steps constitute the user’s deviation from the narrative, in which she takes a detour to the power room to delay the boss’s arrival.

This detour clearly presents a problem for the original narrative. If the boss cannot reach his office, the snooping user will not be caught and the narrative goals for this chapter will go unfulfilled. Specifically, step 300 (*Sabotage*) has an effect of  $\neg\text{working}(\text{Elevator1})$  that threatens the causal link from step 0 (*Initial*) to step 8 (*RideElevator*), which requires that the elevator be operational. Also, step 800 (*Move*) has an effect of  $\neg\text{at}(\text{User}, \text{Hallway})$  that threatens the causal link from step 0 (*Initial*) to step 5 (*Move*), which requires that the user be in the hallway.

## 6.3 Addressing the User’s Alternate Movement

The proactive mediation algorithm selects an exception to handle that is not ordered after any other exception. This is because interventions remove exclusive steps ordered after the exception being addressed, which can potentially remove more exceptions. In this case, step 300 (*Sabotage*) is ordered after step 800 (*Move*), so step 800 (*Move*) is chosen.

Examining the set of failure modes, there are none defined for the (*Move*) operator, so substitution is not possible in this situation. Similarly, there are no directive operators which can assert  $\neg\text{at}(\text{User}, \text{Hallway})$ , so aversion is also not possible. There is no inclusive step that contributes to step 800 (*Move*), so disablement and reordering cannot be performed.

However, accommodation can be performed for this exception. The threatened causal link is removed, allowing the user to move to the power room door. No further plan structure is removed, as step 5 (*Move*) is inclusive, and no additional flaws are added to the plan. This is because step 700 (*Move*) also satisfies the condition  $\text{at}(\text{User}, \text{Hallway})$ . The

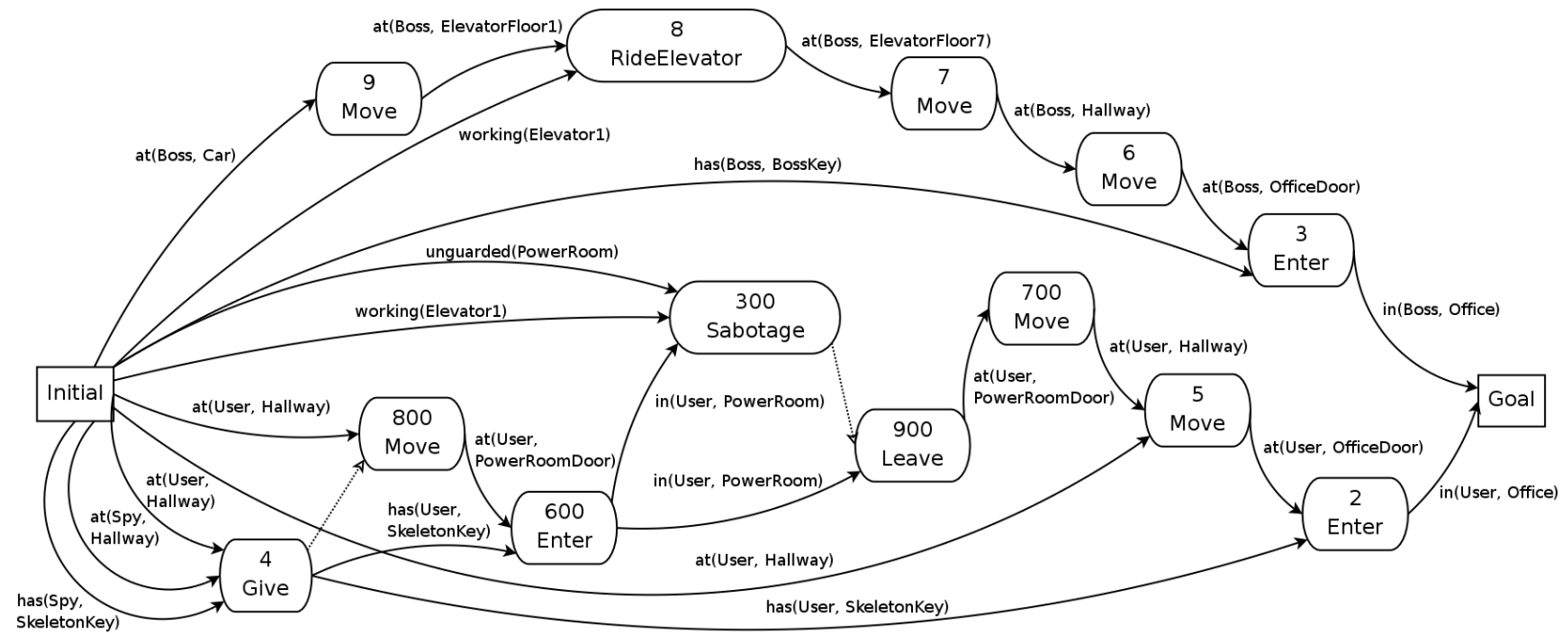


Figure 6.3: The merged plan resulting from the combination of the narrative and recognized plans.

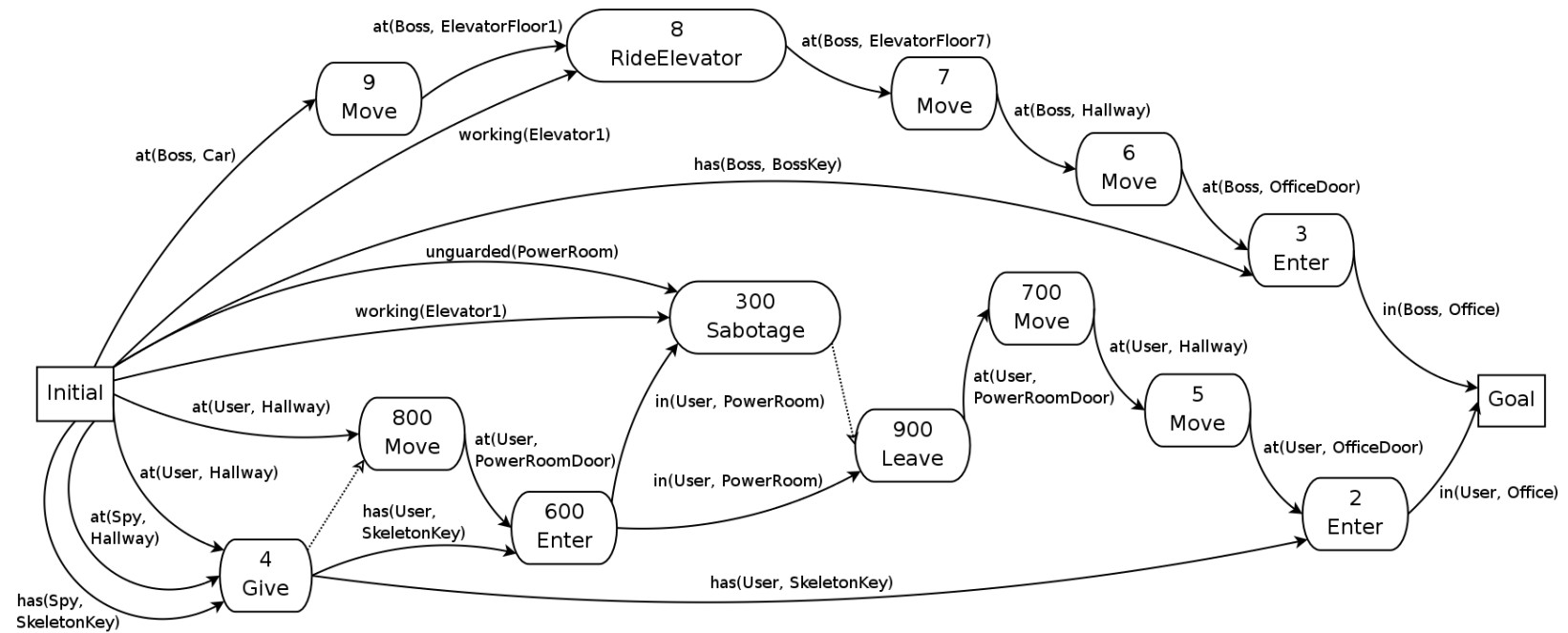


Figure 6.4: The result of avoiding the first exception, step 800 (*Move*) via accommodation.

mediation node containing this accommodated plan and an empty policy table (there have been no substitutions) is then inserted into the mediation space graph as a child of the root node.

## 6.4 Getting the Boss to His Office

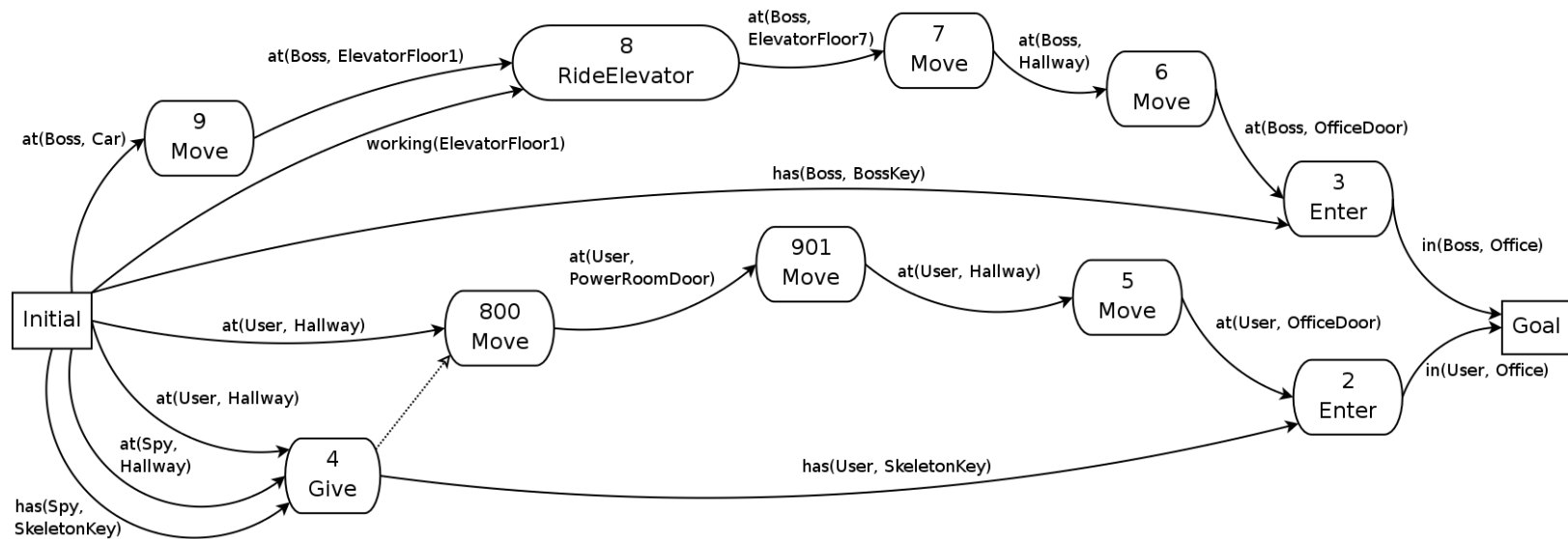
The second exception potentially prevents the boss from riding the elevator, which, according to this narrative plan, prevents him from entering his office and catching the user while she searches for documents. This exception persists because the first exception was avoided via accommodation, which allowed the user to pursue her course of action by moving to the power room. This mediation node is selected for refinement, as it is the only node on the mediation space fringe. The following sections describe the possible responses to step 300 (*Sabotage*).

### 6.4.1 Substitution

The remaining exception in this node is step 300 (*Sabotage*), which has three contributory steps, step 600 (*Enter*), step 800 (*Move*) and step 4 (*Give*). Of the three corresponding operators, only *Enter* has a failure mode definition, *JammedDoor*. There are no effects of *JammedDoor*, so substituting this failure mode will not reestablish  $in(User, PowerRoom)$  nor will it threaten any part of the narrative plan. Therefore, a policy is made to substitute *Enter* with *JammedDoor* between step 4 (*Give*) and step 8 (*RideElevator*).

The resulting mediation node is shown in Figure 6.5, with the policy table entry shown below the plan graph. Replacing step 600 (*Enter*) with a failure mode removes that step from the mediated plan, along with any exclusive steps that follow. This leaves an open precondition of  $at(User, Hallway)$  in step 5 (*Move*), and the complete plan resulting from fixing this flaw coupled with the policy table is one solution in this mediation space.<sup>1</sup>

<sup>1</sup>In Figure 6.5, the causal link with condition  $\neg in(User, PowerRoom)$  is in fact a *preservation link* put in place so that  $in(User, PowerRoom)$  will not be established during the policy interval in the replanning process.



Action	Interval	Response
Enter(User,PowerRoom,PowerRoomDoor,SkeletonKey)	4 - 8	JammedDoor(User,PowerRoom,PowerRoomDoor,SkeletonKey)

Figure 6.5: One final solution using substitution, which replaces an *Enter* action with *JammedDoor*. The text represents the entry in the policy table that enforces this substitution.

### 6.4.2 Aversion

Aversion can be used by instantiating an *inversion step* from one of the directive operators which negates a precondition of *Sabotage* or any of its three contributory steps mentioned previously. There is one inversion step, *StandWatch*, which moves the security guard to watch over the power room, asserting the condition  $\neg\text{unguarded}(\text{PowerRoom})$ . The only ordering constraint applied to the inversion step is that the step comes after the source of the contributory causal link (as there are no other steps in the plan that have  $\text{unguarded}(\text{PowerRoom})$  as an effect).

The resulting plan is shown in Figure 6.6. The inversion step, step 901 (*StandWatch*), has one precondition,  $\text{at}(\text{Guard1}, \text{GuardRoom})$ , which is satisfied by the initial state. Removing step 300 (*Sabotage*) from the plan does not introduce any new flaws, as there are no causal links to any other steps.

### 6.4.3 Disablement

In order to use disablement, a contributory inclusive step must be removed from the mediated plan. In this example, step 4 (*Give*) is the only step that meets those criteria. This step is removed, along with all steps that are ordered after it.<sup>2</sup> This leaves the open precondition  $\text{in}(\text{User}, \text{Office})$  in the goal, which requires replanning to fix. However, the replanning process could put the step  $\text{Give}(\text{Spy}, \text{User}, \text{SkeletonKey}, \text{Hallway})$  back in the plan, or some other step that establishes  $\text{has}(\text{User}, \text{SkeletonKey})$ . To prevent this, Kyudo inserts a preservation link,  $\neg\text{has}(\text{User}, \text{SkeletonKey})$  between step 0 (*Initial*) and step 8 (*RideElevator*)

The resulting plan is shown in Figure 6.7. This plan involves the spy giving the user an alternate key, the *OfficeKey*, which opens the office door, but not the power room door. The user then moves to the office door and enters the office, fulfilling her part of the narrative.

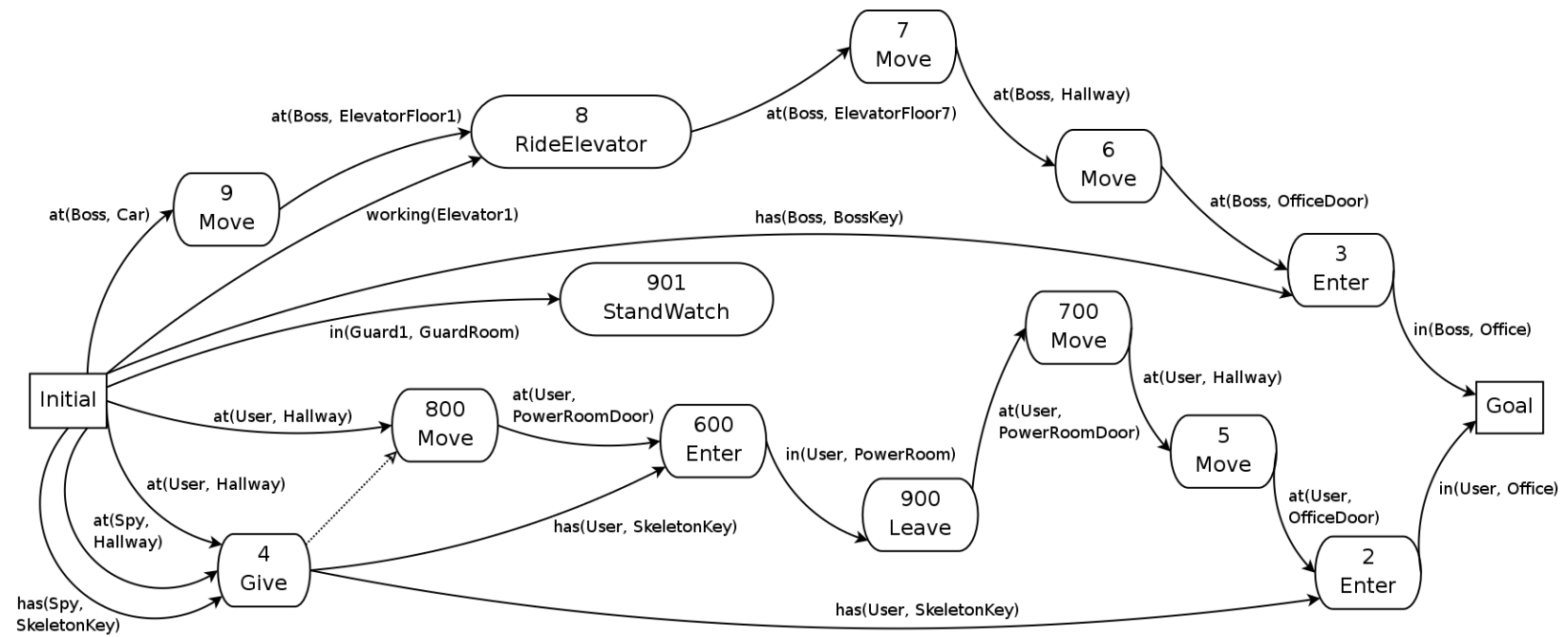


Figure 6.6: One final solution using aversion, placing a guard in the power room to stop the user from sabotaging the elevator.

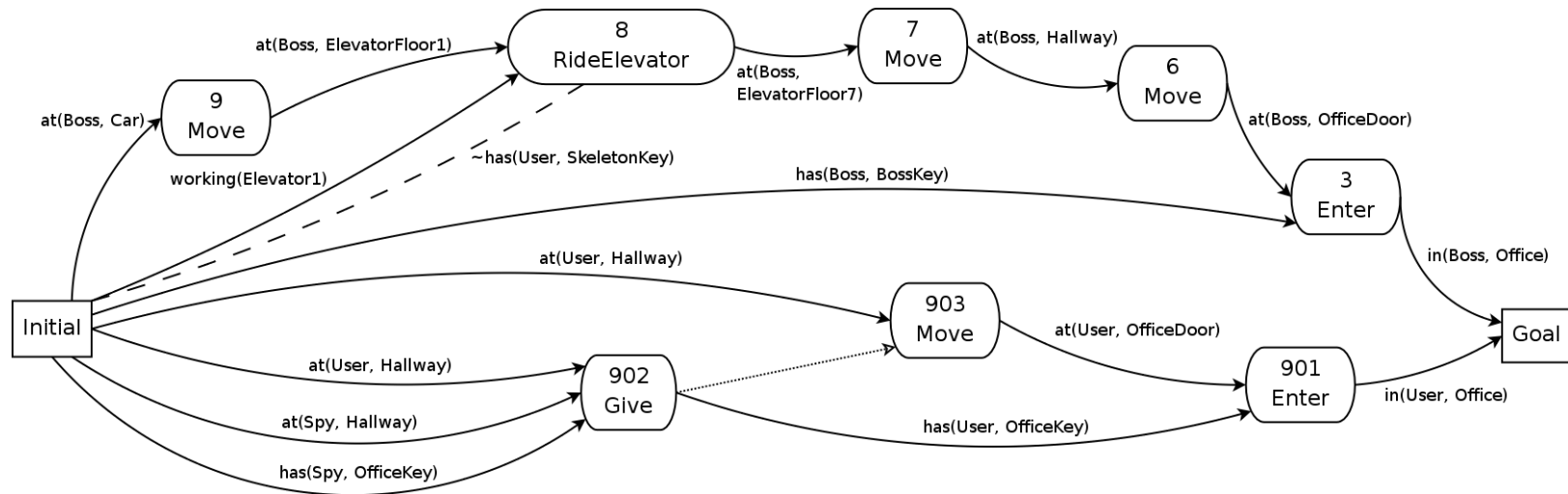


Figure 6.7: One final solution using disablement, directing the spy to give user an alternate key that opens only the boss's office door.

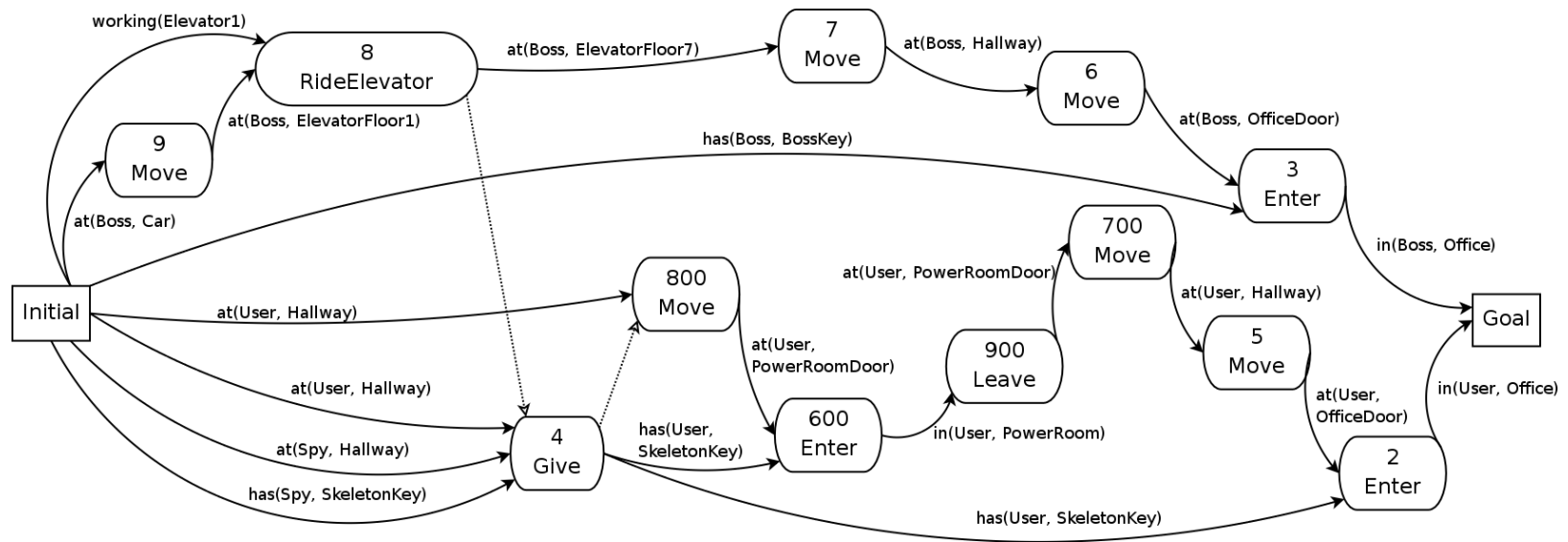


Figure 6.8: One final solution using reordering, directing the spy to wait until the boss is on the seventh floor before giving the key to the user.

#### 6.4.4 Reordering

One of the most straightforward responses to avoid the *Sabotage* exception is to apply a reordering. This places an additional ordering link from the end of the threatened causal link to an inclusive step that contributes to step 300 (*Sabotage*), in this case step 8 (*RideElevator*) to step 4 (*Give*). Adding this ordering ensures that the step 300 (*Sabotage*) does not threaten the causal link (*working(Elevator1)*) by waiting for the boss to arrive on the seventh floor before giving the master key to the user. The resulting plan is shown in Figure 6.8.

#### 6.4.5 Accommodation

Accommodating step 300 (*Sabotage*) allows it to execute, while removing the threatened causal link from the narrative plan. Kyudo also removes step 8 (*RideElevator*) and all steps ordered after it, in order to replan the narrative incorporating the user's recognized steps. A suitable plan is found in which the boss takes the stairs to the seventh floor instead of riding the elevator (perhaps he has a new exercise routine), as shown in Figure 6.9.

### 6.5 Discussion

The solutions discussed in the above example were generated by Kyudo, using Crossbow locally to perform replanning where appropriate. The system was executed with the inputs listed in Appendix A on a Pentium 4 (2.66 GHz) processor with 448 MB of RAM running Windows XP, and the resulting execution times (in seconds) are shown in Table 6.1.

It should be noted that the algorithm explored all five responses for the *Move* step. However, since no suitable applications of aversion, disablement, substitution or reordering could be performed in this case, their execution times were negligible. Using disablement or reordering requires an inclusive step that contributes to the exception, which does not exist in this case, and the contributory check only took one comparison (as the source of the only

---

<sup>2</sup>The only real requirement is that step 4 (*Give*) is removed along with all exclusive steps that are ordered after it. The replanning process may remove other steps, and indeed the default replanner in Kyudo removes all actions ordered after the disabled step.

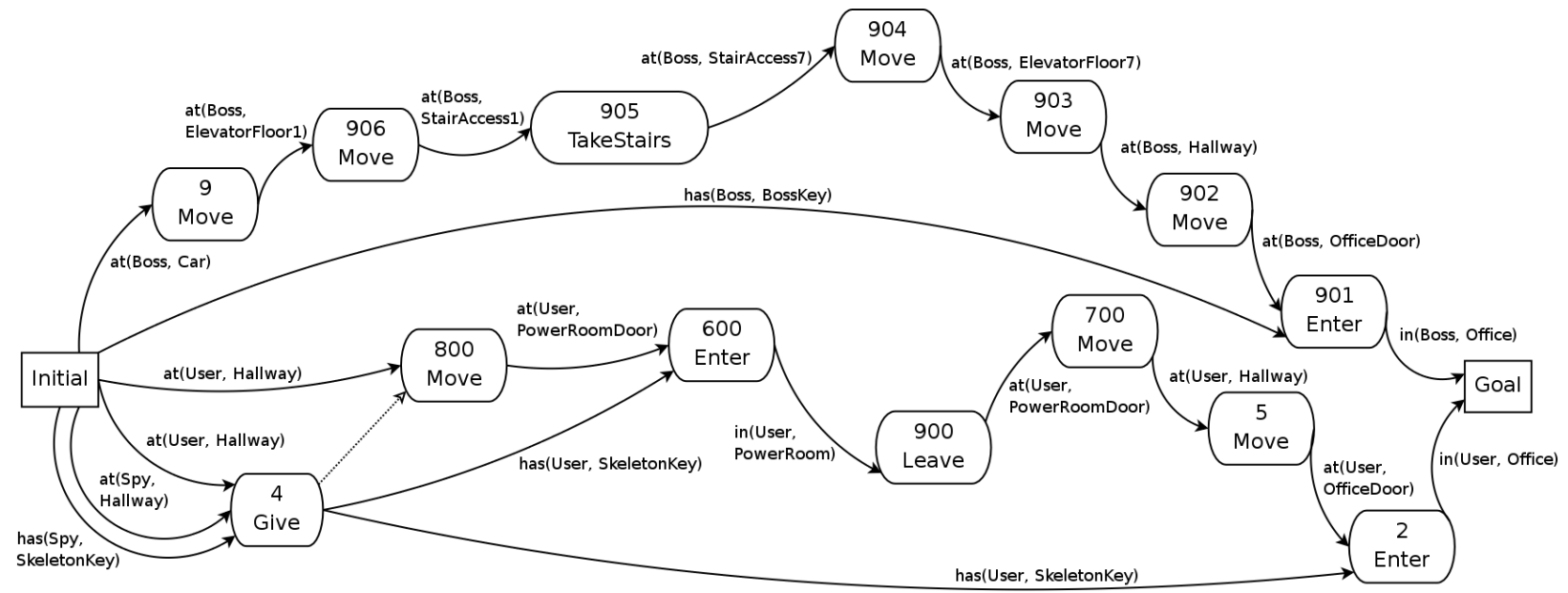


Figure 6.9: One final solution using accommodation, allowing the Sabotage act to be performed.

Response	Time (in seconds)
Accommodation (Move)	0.053410133
Accommodation (Sabotage)	1.165008533
Aversion (Sabotage)	0.0701008
Disablement (Sabotage)	0.063424533
Reordering (Sabotage)	0.0100144
Substitution (Sabotage)	0.0400576

Table 6.1: The average execution time of each exception avoidance in the Corporate Espionage problem (over three trials).

incoming causal link is the *Initial*). The other two responses, substitution and aversion, each required only a look-up in a hash table. The substitution routine found no failure modes defined for the *Move* operator, and the aversion routine found no directive operator that could unify with  $\neg at(Us\text{er}, Hallway)$ . The accommodation for *Move* takes little time because the only modification to the plan is the removal of a single causal link. As stated above, this does not introduce any new open preconditions to the plan, so replanning is not necessary.

Accommodating the *Sabotage* exception takes substantially longer because a large portion of the narrative is taken out to remove the causal link  $working(Elevator1)$ . The execution of the removal routine is proportional to the number of steps in the plan, and its execution time was negligible, but the replanning required to generate a response was significant as a result.

The aversion execution time is similarly dictated by replanning, as the routine to remove extra exclusive steps is proportional to the number of exclusive steps. The aversion routine also includes adding inversion steps to the plan, which is proportional to  $s \cdot d \cdot e \cdot c$ , where  $s$  is the number of contributory steps (plus the exception),  $d$  is the number of directive operators,  $e$  is the average number of effects per directive operator, and  $c$  is the number causal links in the plan. In this case,  $d = 1$  and  $e = 2$ , and there was only one applicable step, *Sabotage* ( $s = 1$ ), which resulted in a negligible execution time. The plan modification in the aversion routine did add an open precondition to the plan,  $in(Guard1, GuardRoom)$ , which is a precondition of the inversion step *StandWatch*. A suitable plan is quickly found by extending a causal link from the *Initial* step to the inversion step.

Disabling the *Give* step essentially removes all of the user's part in the narrative, and must be replanned to get the user into the office. The execution time of the removal

process, as in the previous cases, is negligible, and, as in accommodation, a significant portion of the execution time is spent in the replanning process.

The reordering solution took the least amount of time, executing four times faster than substitution. Reordering will most likely be the fastest response for the vast majority of inputs, as this is the only response that does not perform any replanning. The execution time is proportional to the number of steps that contribute to the exception, in this case three.

Substituting *Entry* with *JammedDoor* involves creating a new entry in the policy table and verifying that the failure mode will not threaten the narrative plan. In this case, *JammedDoor* has no effects, so it can be safely inserted into the policy table ( 0.01 seconds). After removing the exclusive causally dependent steps, only one additional step (*Move*) is required to complete the narrative, taking the remaining 0.03 seconds to generate.

One significant detail about Kyudo's implementation is that the replanning process is terminated once a complete plan is found. Alternate approaches can search the plan space for a maximum number of refinements or a maximum computation time, adding all (or some portion) of the complete plans, along with their corresponding policy tables, to the mediation space. This can provide responses that are more desirable to the author and/or user, but can also considerably affect the run-time performance.

## Chapter 7

# Conclusions

In many interactive environments, a human user is permitted to manipulate the environment in a variety of ways. In a plan-based narrative environment, this manipulation may disrupt the actions of other agents or even the actions that the system intends the user to perform. Proactive mediation expands upon reactive mediation to generate a variety of responses to a user's proposed sequence of actions in the environment. Having a hypothesis about future user actions allows proactive mediation to generate a broader range of responses to user actions that can be temporally distributed over the course of a plan, which allows narrative systems to take a more active role in mediating a user's interaction with the story. In addition, although Kyudo has not been tested on a wide range of domains, the run-times obtained in this study are encouraging. Even when a user's recognized plan contains several exceptions, Kyudo can generate a range of responses in a time suitable for many narrative environments.

There are, however, additional factors to consider in evaluating this approach. Proactive mediation relies on effective plan recognition and can be sensitive to the frequency of change in input from the plan recognition component. In addition, planning is computationally complex; in cases where many proactive responses require replanning, there is no guarantee that an appropriate response can be generated within a reasonable amount of time. However, preliminary consideration indicates that in proactive, plans containing potential exceptions occur relatively rarely compared to the number of actions that a user is likely to perform at any given moment within a story. Most user actions

are consistent or constituent; it is the effects of exceptions on the coherence of the story rather than their frequency that motivates the need to address them. While computation performed by proactive mediation can be costly in some cases, the approach outlined is readily implemented as an anytime algorithm: should proactive mediation fail to generate a response in time to address an exception, reactive mediation can still be used. Similarly, should reactive mediation fail to generate an accommodation in time, intervention (which typically amounts to a straightforward look-up) can be invoked.

## 7.1 Future Work

The current implementation of Kyudo can effectively mediate a user's actions within the context of a narrative plan. However, new questions have been raised, which can serve to guide the future direction of research in this area. The following are potential areas to refine and extend this work:

- One of the inputs to the proactive mediation algorithm is a plan that the human user of the Zocalo system is pursuing, proposed by a plan recognition system. A first step for future work might be to explore the integration of a plan recognition component into Zocalo with the specific role of feeding into Kyudo. There have been many approaches to plan recognition, and selecting and customizing a suitable approach for our needs would allow for further study of the practicalities of mediating various environments, as well as highlight potential areas for refinement and extension.
- While this work defines a mediation space of possible avoidances of harmful actions, it is up to the authors or knowledge engineers of a particular narrative to provide heuristics for exploring that space. Developing tools and representations for these heuristics, as well as performing empirical studies evaluating the effectiveness of responses can aid in efficiently producing desirable solutions.
- The current approach taken when replanning the narrative removes all plan structure that is causally dependent on the original step(s) being removed. While this approach is taken to maximize the chances of finding a suitable plan, it has its drawbacks. First, there is the possibility that the modified plan will be very similar to the original, with only a few small modifications. This can be inefficient, as a large portion of the original

plan is cleared out just to be reinserted by the replanning process. In addition, for very large plans this can be impractical. The cost of replanning can be too great to use in this setting, as it can be critical to put a new narrative plan in place to stop deviant user behavior. Future work could explore more efficient methods of removing plan structure, or incorporating adaptive planning [12] to generate the new narrative.

- When exceptional steps in the recognized plan are identified, only ordering constraints are taken into account. That is, a step is marked as exceptional if it *can* occur during the interval of a threatened causal link in the narrative plan. This can be overly cautious if, for example, the causal link begins much later in the plan and the exceptional step is hypothesized to occur very soon. The use of temporal knowledge in the plan inputs [2] can not only aid in determining the likelihood of exceptions, but also in loosening the “instantaneous” execution requirement of directive operators.
- Restructuring the narrative plan may result in system-controlled agents performing actions that are not clearly motivated. For instance, in the corporate espionage example, the boss has no obvious reason to take the stairs to the seventh floor if his usual routine is to take the elevator. Worthwhile future work would include integration of the proactive mediation component with an intent-driven planner [28] that generates plans where agents’ actions can be understood in terms of their own beliefs, desires, and intentions.

In this thesis, a method for altering a narrative in anticipation of a user’s activity is presented. As one may expect from the corporate espionage example given, one application of this technology would be in the field of interactive entertainment. Video games and virtual reality environments can better adapt a storyline to fit the user’s behavior as it is being played out, engaging the user while maintaining the author’s vision. In addition, proactive mediation could be used in tutoring and training software, such as Steve [26], or Intelligent Help Systems (IHS), such as SmartAidè [23]. Responding to a user as they begin to move toward harmful behavior can not only help to avoid it, but also inform the user as to *why* their intended course of action should be avoided.

# Bibliography

- [1] D. Albrecht, I. Zuckerman, and A. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8(1-2):5–47, 1998.
- [2] James F. Allen. Planning as temporal reasoning. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR'91: Principles of Knowledge Representation and Reasoning*, pages 3–14. Morgan Kaufmann, San Mateo, California, 1991.
- [3] Ruth Aylett. Emergent narrative, social immersion and storification. In *In Proceedings of Narrative and Learning Environments Conference (NILE00)*, 2000.
- [4] Ruth Aylett, Sandy Louchart, João Dias, Ana Paiva, and Marco Vala. Fearnot! - an experiment in emergent narrative. In *Intelligent Virtual Agents, 5th International Working Conference*, pages 305–316, 2005.
- [5] Mathias Bauer. Acquisition of user preferences for plan recognition. In *Proceedings of the Fifth International Conference on User Modeling*, pages 105–112, 1996.
- [6] S. Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.
- [7] M. Cavazza, F. Charles, and S.J. Mead. Planning characters' behaviour in interactive storytelling. *The Journal of Visualization and Computer Animation*, 13:121–131, 2002.
- [8] D. Christian and R. Young. Comparing cognitive and computational models of narrative structure. In *Proceedings of the National Conference of the American Association for Artificial Intelligence*, 2004.

- [9] ECMA. *C# Language Specification*, 2001.
- [10] Michael Fagan and Pàdraig Cunningham. Case-based plan recognition in computer games. In *Proceedings of the 5th International Conference on Case-Based Reasoning*, pages 161–170, 2003.
- [11] Andrew S. Gordon and Nicholas V. Iuppa. Experience management using storyline adaptation strategies. In *Proceedings of Technologies for Interactive Digital Storytelling and Entertainment Conference*, pages 19–30, 2003.
- [12] Steve Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.
- [13] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167–238, 1995.
- [14] J. Laaksolahti. and M. Boman. Anticipatory guidance of plot. *ArXiv Computer Science e-prints*, 2002.
- [15] John E. Laird. It knows what you’re going to do: adding anticipation to a quakebot. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, Montreal, Canada, 2001. ACM Press.
- [16] John E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(3):1–64, 1987.
- [17] Ari Lamstein and Michael Mateas. Search-based drama management. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 2004.
- [18] Neal Lesh. Adaptive goal recognition. In *International Joint Conference on Artificial Intelligence*, pages 1208–1214, 1997.
- [19] M. Lombard and T.B. Ditton. At the heart of it all: The concept of presence. *Journal of Computer-Mediated Communication*, 3(2), 1997.
- [20] B. Magerko and J.E. Laird. Mediating the tension between plot and interaction. In *AAAI Workshop Series: Challenges in Game AI*, pages 108–112, 2004.

- [21] M. Mateas and A. Stern. Architecture, authorial idioms and early observations of the interactive drama façade. Technical Report CMU-CS-02-198, Carnegie Mellon University, 2002.
- [22] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114, San Mateo, California, 1992. Morgan Kaufmann.
- [23] Ashwin Ramachandran and R. Michael Young. Providing intelligent help across applications in dynamic user and environment contexts. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 269–271, New York, NY, USA, 2005. ACM Press.
- [24] M.J. Rattermann, L. Spector, J. Grafman, H. Levin, and H. Harward. Partial and total-order planning: Evidence from normal and prefrontally damaged populations. *Cognitive Science*, 25(6):941–975.
- [25] R.E.Fikes and N.J.Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- [26] J. Rickel and W.L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13:343–382, 1999.
- [27] Mark Riedl, C.J. Saretto, and R. Michael Young. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems*, pages 741–748, 2003.
- [28] Mark Riedl and R. Michael Young. An intent-driven planner for multi-agent story generation. In *Proceedings of the 3rd Autonomous Agents and Multiagent Systems Conference*, pages 186–193, 2004.
- [29] E. Sacerdoti. The nonlinear nature of plans. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.

- [30] Peter William Weyhrauch. *Guiding interactive drama*. PhD thesis, Carnegie Mellon University, 1997. Chair-Joseph Bates.
- [31] R. Michael Young. Notes on the use of plan structures in the creation of interactive plot, 1999.
- [32] R. Michael Young. Creating interactive narrative structures: The potential for ai approaches, 2000.
- [33] R. Michael Young, M. Pollak, and J.D. Moore. Decomposition and causality in partial-order planning. In *Proceedings of the Second AI Planning and Scheduling Conference*, pages 188–193, 1994.
- [34] R. Michael Young, Mark Riedl, M. Branly, R.J. Martin, and C.J. Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1:52–70, 2004.

## Appendix A

# Example Inputs

Goal State
in(Boss, Office)
in(User, Office)

Table A.1: Goal State of the Problem

<b>Initial State</b>	
accesses(StairAccess1, MainStairwell)	accesses(StairAccess7, MainStairwell)
at(Boss, Car)	at(Spy, Hallway)
at(User, Hallway)	controls(ElevatorController1, Elevator1)
has(Boss, BossKey)	has(Spy, OfficeKey)
has(Spy, SkeletonKey)	in(Guard1, GuardRoom)
in(ElevatorController1, PowerRoom)	isaArea(Office)
isaArea(PowerRoom)	isaArea(GuardRoom)
isaCharacter(Boss)	isaCharacter(Spy)
isaCharacter(User)	isaGuard(Guard1)
isaController(ElevatorController1)	isaDoor(OfficeDoor)
isaDoor(PowerRoomDoor)	isaElevator(Elevator1)
isaElevatorLocation(ElevatorFloor1)	isaElevatorLocation(ElevatorFloor7)
isaKey(BossKey)	isaKey(OfficeKey)
isaKey(SkeletonKey)	isaLocation(ElevatorFloor1)
isaLocation(ElevatorFloor7)	isaLocation(Hallway)
isaLocation(OfficeDoor)	isaLocation(PowerRoomDoor)
isaLocation(StairAccess1)	isaLocation(StairAccess7)
isaStairAccess(StairAccess1)	isaStairAccess(StairAccess7)
isaStairwell(MainStairwell)	leadsTo(OfficeDoor, Office)
leadsTo(PowerRoomDoor, PowerRoom)	nextTo(Car, ElevatorFloor1)
nextTo(ElevatorFloor1, StairAccess1)	nextTo(ElevatorFloor7, Hallway)
nextTo(Hallway, OfficeDoor)	nextTo(Hallway, PowerRoomDoor)
nextTo(PowerRoomDoor, Hallway)	nextTo(StairAccess7, ElevatorFloor7)
opens(BossKey, OfficeDoor)	opens(OfficeKey, OfficeDoor)
opens(SkeletonKey, OfficeDoor)	opens(SkeletonKey, PowerRoomDoor)
services(Elevator1, ElevatorFloor1)	services(Elevator1, ElevatorFloor7)
unguarded(PowerRoom)	working(Elevator1)

Table A.2: Initial State of the Problem

Operator	Constraints	Preconditions	Effects
Enter	isaArea(?room) isaCharacter(?char) isaDoor(?from) isaKey(?key) isaLocation(?from) leadsTo(?from, ?room) opens(?key, ?from)	at(?char, ?from) has(?char, ?key)	$\neg$ at(?char, ?from) in(?char, ?room)
Give	isaCharacter(?getter) isaCharacter(?giver) isaKey(?key) isaLocation(?location)	at(?getter, ?location) at(?giver, ?location) has(?giver, ?key)	$\neg$ has(?giver, ?key) has(?getter, ?key)
Leave	isaArea(?room) isaCharacter(?char) isaDoor(?to) isaLocation(?to) leadsTo(?to, ?room)	in(?char, ?room)	$\neg$ in(?char, ?room) at(?char, ?to)
Move	isaCharacter(?mover) isaLocation(?from) isaLocation(?to) nextTo(?from, ?to)	at(?mover, ?from)	$\neg$ at(?mover, ?from) at(?mover, ?to)
RideElevator	isaCharacter(?char) isaElevator(?elevator) isaElevatorLocation(?from) isaElevatorLocation(?to) services(?elevator, ?from) services(?elevator, ?to)	at(?char, ?from) working(?elevator)	$\neg$ at(?char, ?from) at(?char, ?to)
Sabotage	controls(?controller, ?elevator) in(?controller, ?room) isaArea(?room) isaCharacter(?saboteur) isaController(?controller) isaElevator(?elevator)	in(?saboteur, ?room) unguarded(?room) working(?elevator)	$\neg$ working(?elevator)
TakeStairs	accesses(?from, ?stairwell) accesses(?to, ?stairwell) isaCharacter(?char) isaStairAccess(?from) isaStairAccess(?to) isaStairwell(?stairwell)	at(?char, ?from)	$\neg$ at(?char, ?from) at(?char, ?to)

Table A.3: Domain Operators

<b>Failure Mode (Operator)</b>	<b>Constraints</b>	<b>Preconditions</b>	<b>Effects</b>
FailedSabotage (Sabotage)	controls(?controller, ?elevator) in(?controller, ?room) isaArea(?room) isaCharacter(?saboteur) isaController(?controller) isaElevator(?elevator)	in(?saboteur, ?room) unguarded(?room) working(?elevator)	[none]
JammedDoor (Enter)	isaArea(?room) isaCharacter(?char) isaDoor(?from) isaKey(?key) isaLocation(?from) leadsTo(?from, ?room) opens(?key, ?from)	at(?char, ?from) has(?char, ?key)	[none]

Table A.4: Failure Modes

<b>Operator</b>	<b>Constraints</b>	<b>Preconditions</b>	<b>Effects</b>
StandWatch	isaArea(?room) isaArea(?from) isaGuard(?guard)	in(?guard, ?from)	$\neg$ unguarded(?room) in(?guard, ?room) $\neg$ in(?guard, ?room)

Table A.5: Directive Operators