ENABL: A Modular Authoring Interface for Creating Interactive Characters

April Grow

Abstract

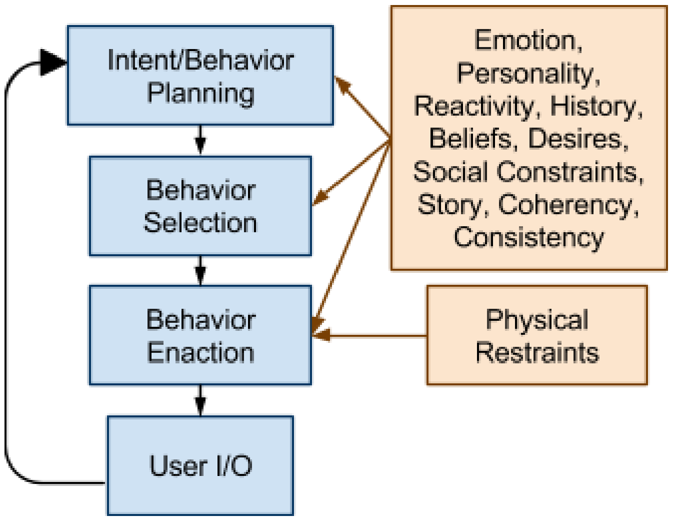
The authoring of interactive digital characters is a complicated and multidisciplinary task which requires some level of expertise in psychology, artificial intelligence, computer graphics, as well as many other fields depending on the purpose of the agent. Demand for high-quality agents is enormous: current research involves interactive agents for many topics including education, medicine, human caregiving, interactive drama, and video games. However, experts able to author these agents are few in number, and supply is falling far short of demand. This work aims to develop a publicly releasable authoring environment for an existing reactive planning behavior language (ABL: A Behavior Language), which significantly reduces the authoring complexity and burden in creating interactive digital characters. By reifying existing coding practices and patterns by expert authors into an authoring environment, authors with some programming background will be able to create interactive characters for any field or subject matter much more quickly and easily than with the programming language alone. Ultimately, the breadth and depth of interactive experiences involving agents will vastly increase, which may revolutionize current approaches to education, video games, and many other fields.

Introduction

Creating robust, believable, and interactive characters is extremely challenging. Because we experience and interact with human-like behavior throughout our daily lives, every person has some idea of what is (or is not) acceptable behavior. We are all equipped to critique artificial behavior, but designing artificial behavior is much more difficult. Artificial intelligence (AI) is an entire field devoted to modeling this behavior through a multitude of approaches and purposes, and there are many challenging AI problems at the core of creating interactive characters (natural language processing, navigation, and reasoning, for example).

Let us examine an example dramatic character performance pipeline to better understand the challenge that authoring the pipeline entails. An agent, with sensory information from its environment, uses some mechanism to figure out what to do (the focus of this document will be on reactive planning as this method). What the agent is capable of doing is represented as some possibility space, such as a behavior tree. The behavior(s) that are chosen by the planner are realized in some 2D or 3D engine, likely via a library of canned animations. Finally, the user, the environment, or another agent offers some input to the agent's sensory system and the cycle begins again. The author of the agent must guide the agent(s) through this pipeline while attempting to juggle a large set of behavioral expectations for the agent (which we describe in detail later as believability and dramatic constraints). The whole path can be summarized in Figure 1.

We define the term *authorial complexity* as the challenge of creating each behavior, which potentially interfaces with every other previous behavior. The collection of all these behaviors together represents the *authorial burden*, a concept used to describe the total authoring task of the whole agent. As the agent becomes more robust with richer world state, agent states, longer and more complex histories, and a wide range of possible animations, incorporating a new intent or behavior becomes extremely complex, as that new behavior may interface or interact with all the previous behaviors. The author may choose not to meet this challenge by actively choosing not to interface with existing behaviors, fail to meet the challenge by not *deeply* interfacing with the existing elements, or fail by interfacing with them in nonsensical ways.


Intent/Behavior Planning
Behavior Selection
Behavior Enaction
User I/O
Emotion, Personality, Reactivity, History, Beliefs, Desires, Social Constraints, Story, Coherency, Consistency
Physical Restraints
Figure 1: High-level summary of agent architecture elements that the author must keep track of.

To reduce the authoring challenge, the combinatoric interface of intents, behaviors, and animations must be made more manageable, structured, and modular. The author proposes to analyze and break down the authoring challenge into separate modules and examine how those modules relate to the different aspects of agent authoring. The results of this analysis will then inform and guide the creation of a concrete authoring interface for ABL that addresses, supports, and alleviates as many of the authoring challenge components as possible.

Primary Research Question

- Can modularity reduce the authorial complexity of creating dramatic, embodied, and interactive agents?

Authorial complexity is seen as the primary barrier standing in the way of creating more dramatic agents, as well as more agents in general. Whether the complexity takes too much time to debug and update, or whether the complexity simply becomes logically impenetrable, both are unacceptable problems that must be overcome. The modular approach proposed and planned in this document aims to drastically shorten the debugging time and reduce the points of interaction between behaviors to a tractable level.

Secondary Research Questions

- How can dramatic agent authorial complexity be reduced through modularity?

The nature of the authorial burden is directly tied to the task the author has to complete. If the author must create or interweave dramatic story management, agent planning, behavior selection, and animation, trying to create and debug them all simultaneously is a mess (although they will all be connected in the final product). Visualization modules proposed in this document chop up debugging into specific, targeted tasks that each have a focused interface.

The task-focused modules allow us to create targeted tools for the author's use, and we propose them as the most productive approach to answering *how* modularity can reduce complexity.

- What benefits do working with modular authorial patterns and interfaces provide to authors?

Our evaluation proposes a slew of iterative user studies to judge the possible benefits to debugging time, author's mental models of the system, and agent complexity. Although authoring experiences have been notoriously difficult to describe, let alone evaluate, we look toward decision trees applied to interactive narrative as a starting place for qualitative and quantitative evaluation. We also plan to apply task-based usability testing to the interfaces in order to judge how novice, intermediate, and expert authors may benefit from the authoring interface (also referred to throughout this document as an authoring tool).

The distinction of embodied, interactive, and dramatic characters is crucial to make, as it sets a relatively challenging lower bar of acceptable output from both the foundational AI architecture and the authoring support tool. By "dramatic," we ascribe to Mateas' view that dramatic agents extend the concept of believable agents with the addition of story considerations. The resulting characters must be able to react to other characters (interactive) and perform the authored behaviors in a perceptible manner (embodied) in order to satisfy the author. The authoring tool makes no claims about ensuring the dramatic quality of the authored behaviors, as that level of authoring is in the hands of the human programmer and their scenario goals. However, the characters must be capable of dramatic performance, including such elements as personality, emotions, reactivity, social considerations, and story. While interactivity frames the character's potential inputs and embodiment frames its outputs, dramatic performance provides the author the means to express their character's goals as richly as possible. It is this focus on rich, complex, dramatic agents that have led us to select ABL as the target architecture on which to build our authoring interface.

First, the proposed research will be positioned in the space of existing AI agent architectures and a more detailed rationale will be given for the author's choice of ABL as a foundation. Previous projects involving ABL will also be examined for coding idioms, implicit rules, and best practices that can be reified within an authoring environment. In the proposed work, general authorial patterns will be extracted from the related and previous work. To demonstrate these patterns, the authoring environment will then be presented in great detail in the context of ABL, assembling all the presented guidelines into an actionable piece of software. The following evaluation section will use previous analysis of complex behavior-tree -based analysis to examine the outputs of a long, on-going, iterative plan for many usability testing case studies. Finally, a schedule will outline the future creation of the authoring tool and the planned case studies within the span of the author's career as a PhD candidate.

Contribution

The general contribution of this work is a detailed analysis of the dramatic agent authoring process, proposals to aid the authoring process, as well as a modular authoring design pattern that can be applied to any agent architecture (and likely other complex architectures). The proposals and design pattern are supported by previous work examining expert author best practices from multiple architectures, not just ABL. To verify the proposals and design pattern, they will be applied to the ABL framework. Detailed documentation of the application process will be published as a guide for other architectures looking to modularize their authoring approach. Evaluation methodologies will then be applied from interactive narrative to agent architectures, a novel application strategy which will result in a much more detailed analysis of agent quality than previously seen for dramatic agents.

Related Work

Virtual Characters

Because virtual characters have such a broad definition and are used in so many applications, the applicable useful research is very broad. The following sections help both focus this advancement's use of the term "virtual character" and its related keywords, while also learning lessons from other closely related fields (such as robotics). These fields contain many areas of overlap in their design and authoring challenges, and thus contain many relevant lessons, but their subject matter may not fit exactly with the type of agents we are focusing on in this paper.

Reactive, Interactive, Dynamic, Procedural, Social, Autonomous

There are a number of similar keywords that are used to represent the idea of agents, supported by some form of AI, that respond to input with some detectable or discernable output. Humans are enchanted by behavior that appears to recognize and acknowledge us, especially in inanimate machines. The *Senster* was an early example of this reactivity, and it elicited awe from its beholders with very simple, yet life-like, behaviors (Ihnatowicz 1986). ELIZA, a famous (or infamous) conversational agent, captured the hearts and minds of many users despite its simplicity, much to the dismay of its author, Weizenbaum (Weizenbaum 1979).

However, expectations in the quality and complexity of these agents have progressed much more quickly than their technology. Compare a quick survey of the Believability section below with the current state of agent AI in popular media, and the huge gulf between them becomes apparent. In Figure 2, we see an example of agent AI horribly breaking immersion because the player's side-kick was not given the intelligence to operate within the confines of different combat scenarios. Instead, the rules of the world simply do not apply to her: her noisy actions elicit no response from the sound-activated enemies. The goal of this document is to help begin to bridge the gap between expectations and technology. Authoring more behaviors that maintain world coherence, for example, would then not be such a daunting task. In order to bridge the gap of agent AI and audience expectations, it is a fundamental requirement for the agents in this document to be interactive with other agents and humans.

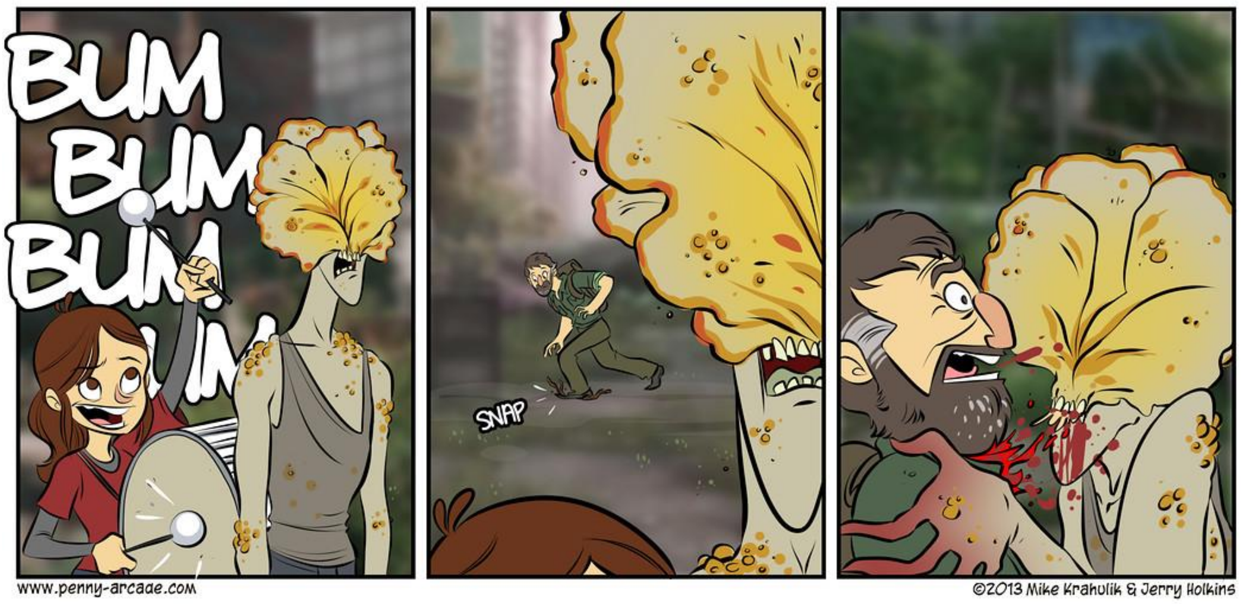


Figure 2: A comic by Penny Arcade about agent AI breaking immersion in *The Last of Us*. Enemy "clickers" are responsive to noise, but your sidekicks will blather on and make a ton of noise, eliciting no response. Other example examples of glaringly broken AI include agents running into walls or standing up out of cover (or in the way of the player) in a fire fight.

Embodied

This paper focuses on virtual agents with virtual bodies, but "embodied" applies to agents that have been given non-virtual bodies as well, and follows the interactionist AI school of thought (Agre 1997). Agre's "mutual constraint between machinery and dynamics" is a powerful design concept that helps authors constrain and conceptualize the scenario they are authoring:

> *For example, an agent that always puts its tools back where they belong my need simpler means of finding those tools than an agent that leaves them lying about.* (Agre 1997)

Embodied agents help restrain the authoring of behaviors to those that are useful in the immediate sense and contextualized by their environment.

Rea (Cassell et al. 1999), Façade (Mateas & Stern 2005), and Ada and Grace (Swartout et al. 2010) are examples of embodied agents created for different purposes. Rea is an anthropomorphized interface to a real estate database who focuses on one-on-one interaction with a single user. Ada and Grace are a pair of agents that interact with each other and potentially many humans in order to engage visitors at the Museum of Science in Boston and "increase their knowledge and appreciation of science and technology" (Traum, et al. 2012). Façade includes agents Grace and Trip enacting a single-act drama about marriage troubles in which the user has the opportunity to help them keep their marriage intact (Mateas & Stern 2005). The environments of these agents help build the user's expectations of the agent's behaviors.

Intelligent

Deciding or determining whether something has intelligence in the context of AI has been a tricky question since the Turing test (Turing 1950). The Turing test operationalized intelligence as the system being indistinguishable from a human in a decontextualized conversational context. In contrast, we are interested in a human interactor, the user, being able to ascribe intelligence to the agent in a particular, artistically heightened, context.
The nature and depth of an agent's interactive components may momentarily give the illusion of intelligence, which Noah Wardrip-Fruin has called *the ELIZA Effect* (Wardrip-Fruin 2009). However, the *ELIZA Effect* illusion cannot be sustained by simple or overly transparent forms of AI. What authored agents should strive to achieve is instead Wardrip-Fruin's *SimCity Effect*: the agent supporting incremental exploration of its behavior model by the player (Wardrip-Fruin 2009). This Effect would help make agent behaviors appear logical and intelligent to the player.

Believable

After the illusion of intelligence faded from early agents, interested parties from a wide variety of disciplines attempted to prevent future illusions from falling quite so quickly by making agents more *believable*. There are many opinions on what makes up a believable agent, which are summarized in the following Table 1, roughly categorized by high-level concepts often seen among many authors.

Concept	Flavor & Proponent	Details, Examples, and Argument
Emotion	Consistency & variability (Ortony 2002)	Agents should make a variety of consistent emotional responses to stimulation.
	Appropriately timed & clearly expressed (Bates 1994)	Agents should perform behaviors that express emotional states (Bates, Loyall, & Reilly 1994). Expressions of these emotions should be true to the character's Personality (Loyall 1997)
	Empathy (Hayes-Roth & Doyle)	Bodily changes that follow the perception of an exciting fact. The feelings the body undergoes from the result of change.
Personality	Piecemeal traits (Ortony 2002)	Contributes to coherence, consistency, and predictability in emotional reactions and responses.
	Personality-Rich Agents (Reilly & Scott 1997)	A high-level term that describes how emotions, agent competence, quirks, relationships, and attitudes should vary between agents to make them feel unique.
	Expressing Individuality or Characteristics (Loyall 1997)	Captures the idea of "character" or "individuality" when describing memorable characters in other media.
	Persona (Hayes-Roth & Doyle)	Multi-faceted individual qualities that affect not just an agent's function, but the performance of their function.

	Recognizable (Perlin & Goldberg 1996)	Perlin & Goldberg specifically mention that agents do not to be *realistic*, but their personalities should be recognizable and consistent.
Reactive/ Responsive	Situated Liveliness (Lester & Stone 1997)	Agents show that they are alert and perceive the world around them.
Self Motivation	Proactive Engagement (Loyall 1997)	Agents don't *just* react/respond to stimuli, but also engage in action "of their own accord."
	Illusion of Life (Loyall 1997)	Agent's motivated actions must appear to be driven by logical goals, bounded by physical and social constraints.
	Motivational State (Blumberg 1996)	"Convey intentionality and motivational state in ways we intuitively understand"
Change with Experience	Robustness (Reilly & Scott 1997)	Agents able to stay "in character" throughout social interactions and history with other agents.
	Growth (Loyall 1997)	Over the course of media, characters change, even if just arbitrarily. These are long-term reactions to stimuli that are still in-line with the agent's personality.
	Adaption (Blumberg 1996)	"Learning new strategies to satisfy goals."
	Remembering (Hayes-Roth & Doyle)	Internal changes result from experience, which lead to observable changes in future behavior
Social	Relationships & Attitudes (Reilly & Scott 1997)	Wildly fluctuating expressions of social state depending on the interactions between agents. (Ex: hatred, friendship, trust)
	Roles and Constraints (Reilly & Scott 1997)	Definitions of socially acceptable behavior and how to handle deviants of these expectations, dependent on social status.
	Social Relationships (Loyall 1997)	Necessary when multiple agents interact, especially over time. Is expressed via emotions and behaviors, and colored by personality.
	Social Relations (Hayes-Roth & Doyle)	Contextualizes and informs interactions with others. Expresses social status, authority, and roles, even inconsequential actions. Can apply to non-humans, such as pets.
Predictable	Predictability (Ortony 2002)	Ortony suggests that users should be able to expect and predict how an agent *should* act (See Personality Entry)
	Consistency (Loyall 1997)	All previous entries of emotions, reactions, motivated behaviors, social interactions, etc must be consistent with the character's personality, history, and relationships.
	Idiosyncratic & Appropriate (Hayes-Roth & Doyle)	Inconsistencies in human behavior and the idiosyncrasies of individuals draw our interest. We should expect appropriate and coherent behavior, but not be able to predict patterns.

Coherence	Contextuality, Continuity, & Temporality (Stone & Lester 1996)	Behaviors must be contextualized by the agent's environment and maintain consistency with all the above dimensions, especially history. Behaviors should be visually coherent to the user.

Table 1: A summary of concepts and elements that different authors have suggested are necessary for a believable agent. Thanks to (Gomes et. al. 2013) and (Isbister & Doyle 2002) for the help in aggregating some of these.

Certainly we'd love to accomplish all of these goals with authored agents. For the purposes of this document, we are not claiming that agents authored by these frameworks are more believable, or objectively satisfy any of these descriptions. Our goal is to support a framework that is capable of expressing whatever of these believability metrics the author wishes to strive for. Isbister & Doyle observes: "Choices about the appearance, personality, and behaviors of the agent are frequently made on the basis of an introspective examination of personal preferences*…*" (Isbister & Doyle 2002). Expressed another way, the author should have the power to express whatever personal preferences they desire, because their authoring choices will largely stem from those personal preferences. The brunt of the work required to create a believable character falls to the author, not the system, after all. The system should be capable of expressing a believable character, and thus support as many of these facets of believability as possible.

So how can an authoring tool support these facets of believability? One approach is to build theories of emotions, personalities, moods, and social structures into the architecture. FAtiMA is one such example, and it is built upon the OCC Theory of Emotion (Dias, Mascarenhas, & Paiva 2011). However, the architecture adheres to the theory so strongly that the author can no longer choose to follow a different theory: the affective state and appraisal model are inescapable. The authors of FAtiMA have moved toward a more modular approach to building additional systems on top of the FAtiMA Core, but what makes FAtiMA FAtiMA is the OCC Theory of Emotion.

Another approach to support authoring believable characters is to make the entire system modular: one module works off the OCC theory and another supports Paul Ekman's fundamental six emotions (Ekman & Friesen 2003); or one module uses the Big 5 personality model (Digman 1990) while another uses Myers-Briggs' types (Myers 1962). So long as there are clear dependencies and interface points defined (as with any code library), an agent author could pick between them. Ultimately, all of these potential agent features should be modularized in this fashion, which is why part of the propose work of this document is a library of modular behaviors.

Dramatic

As stated in the introduction, the virtual characters must be capable of dramatic performance, and, wherever possible, the authoring system should support the author in realizing their dramatic character and scenario designs. In the style of Mateas and Bates, interactive dramatic performance in the context of this document is the combination of interactive freedom and a flexible, compelling, dramatic structure (Bates 1992; Mateas 2002). More concretely, if you take the last table from Believability and add in an additional section related to story, that new table would represent elements of a dramatic agent (Table 2).

Concept	Flavor & Proponent	Details, Examples, and Argument
Story	Premise (Egri, 1960)	"a proposition antecedently supposed or proved; a basis of argument. A proposition stated or assume as leading to a conclusion" (Egri, 1960). The theme, goal, root-idea of the plot that all action must serve or progress in some way.
	Plot Points	"important moments" in a story (Weyhrauch 1997). Makes up the dramatic arc, approximately, or a "plot graph" (Kelso, Weyhrauch, and Bates 1993).
	Dramatic Beat	"the smallest unit of dramatic action" (Reidl & Stern 2006 summarizing McKee 1997). In Façade, these were story pieces on the scale of tens of seconds (Mateas 2002).
	Dramatic Arc (Aristotle 330 BC)	A pattern of rising and falling tension as issues are resolved or raised along the course of a story.

Table 2: A summary of points made by Mateas in the Oz-Centric Review of Interactive Drama and Believable Agents (1999) which was rehashed in his dissertation (2002).

It just so happens that projects involving dramatic agents tend to include some form of drama manager (Mateas 1999; Mateas & Stern 2005b). We don't intend to imply that every AI agent architecture should have a dedicated drama management system, but that some higher-level controlling or organizational concepts helps authors modularize their behaviors and create richer scenarios. Story is simply another module authors should be able to employ in service to their agents or scenario design. Examples of these higher-level organizational concepts in the context of ABL can be seen in the later Idioms section in the related work as Managers (Weber et. al. 2010) and higher-level beat behavior organization (Mateas & Stern 2004). A full list of planned supported idioms can be found near the end of the proposed work section (Table 4). From this point onward, whenever we refer to "dramatic agents," we mean agents that attempt to implement (or plan to attempt to implement) the elements of believable and dramatic agents outlined in the previous two Tables (1 & 2).

Design

Having the intent to make an awesome dramatic agent is all fine and good, but whether the agent actually fulfills these goals is reliant upon the author designing an agent that has all these capabilities. We have briefly described ways to support the design of elements related to believability and drama, but other designers have taken different approaches to general agent design. For example, Sosa and Gero create and use a *design situation* methodology to organize individual and cultural possibilities for a scenario (Sosa & Gero 2003). Shapiro et. al., in their social simulation, used improv and live action to help conceptualize details of their scenario for their graphics and AI authors (Shapiro et. al. 2013). Belief-Desire-Intention (BDI) agents have a design methodology of satisfying the categories of belief, desire, and intent for their agents in order to be fully realized (Rao & Georgeff 1995). Having a quantifiable checklist of authorial tasks grounded in theory can be a useful design approach. Creative requirements engineering is another approach to aid technical brainstorming of agents and their environments (Maiden, Gizikis, & Robertson 2004).

There are additional constraints on the scenario based on its purpose and goals as well, usually made to demonstrate the unique features of the parent architecture or to satisfy more rigorous agent requirements. Wooldridge and others are fond of using mathematical formalisms and logic to describe agents and their environments such that properties of their performance can be deduced and verified (van der Hoek & Wooldridge 2012). However, novice or intermediate authors are not likely trained in formal logic, and so exposing this level of system detail would likely confuse authors and interfere with their creative process. Alternatively, if these formalisms were placed “under the hood” of the authoring tool, they could help the authoring tool reason about the authored agent state space in order to find logical failures or visualize inconsistencies with the author’s mental model (Wooldridge & Dunne 2005). Some of our proposed work regarding offline analysis of an author’s behavior dependency tree and play traces are directly inspired by logical reasoning.

Joanna Bryson designed a developmental methodology specifically for aiding authors in the agent design process, entitled Behavior Oriented Design or BOD, and used it as a governing philosophy in designing the POSH system (Bryson 2003). Inspired by Object-Oriented Design (OOD), BOD explicitly supports a rapidly iterative design cycle and helps authors decompose high-level concepts into concrete behaviors. These concrete behaviors are categorized into three major categories: action patterns (a sequence of primitives that always follow each other), competences (depend on context to manage preconditions), and drive collections (which are constantly being checked). These categories roughly represent (hierarchically) low-, medium-, and high-level behavior selection mechanisms respectively, and examining the complexity of each element is a quick way to determine when authoring complexity is becoming too overwhelming. The clean separation of tasks, behavior hierarchy levels, and approach to implementation is an extremely attractive philosophy to follow. We aim to espouse this philosophy and use it to guide our authors in module design, creation, and iteration in our proposed work and in setting up our evaluations.

Dynamic Reactive Planning

Behavior Trees

Given the agent specifications and design goals described above, behavior trees have historically given the most bang for the authoring buck. Originally introduced by R. Geoff Dromey and described as Genetic Software Engineering (Dromey 2001), he redefined them as Behavior Trees to avoid confusion with genetic programming (a procedural content generation technique):

> *A Behavior Tree is a formal, tree-like graphical form that represents behavior of individual or networks of entities which realize or change states, make decisions, respond-to/cause events, and interact by exchanging information and/or passing control… They provide a direct and clearly traceable relationship between what is expressed in the natural language representation and its formal specification.* (Dromey 2003)

Given the previous squishy descriptions of what makes up a believable agent and how complicated fitting all those features into one agent can be, using behavior trees rooted in the expression of natural language makes sense: keep the authoring as simple as possible because the constructs are super complex!

The behaviors in behavior trees can be abstract, concrete, or anything the author desires, and the architecture of behavior trees is very scalable compared to most other decision-making mechanisms (Lindsay 2010; Powell 2010)]. Wen & Dromey built out functional behavior trees and a software tool, BECIE, to simulate a Universal Turing Machine and test its scalability (Wen & Dromey 2009). Although behavior trees originated as a formal graphical modeling system, they have since been appropriated by many academics and industry professionals for agent design in less formal manners (Champandard 2007, Simpson 2014).. Example comercial instances of confirmed, large-scale behavior trees include *Halo 2* (Isla 2005) and *Spore* (Hecker 2009).

Hap

Hap is a reactive, goal-directed agent architecture developed in the Oz project (Loyall & Bates 1991). It works around an “active plan tree” (which predated the behavior tree terminology but functions similarly) with nodes containing goals and subgoals for completion. The dynamism and reactivity of Hap’s plan tree means that the tree itself changes shape, allowing the subgoals to be reused and the agents to be more reactive (a key aspect of agent believability mentioned above) (Georgeff, Lansky & Schoppers 1987; Georgeff & Lansky 1987; Firby 1987). Loyall presented Hap in his dissertation work as “an architecture specifically designed to support the requirements and expression of believable agents,” including support for agent and human interaction, animations, arbitrary (read: potentially artistic) emotions/personalities, and other aspects of believability all in one architecture (Loyall 1997).

ABL

ABL, A Behavior Language, is an extension of the Hap language, and was the agent architecture behind one of the most influential and extensive examples of interactive dramatic agents research: *Façade* (Mateas & Stern 2003; 2004; 2005; 2005b). ABL extended Hap with more generalized connections between actions and sensors, atomic behaviors, meta-behaviors, goal spawning, support for joint behaviors, among other things (Mateas 2002). In particular, ABL carried on Hap’s legacy of supporting believable characters and their ease of authoring, all while enhancing agent complexity and expressivity, as well as multi-agent capabilities (Mateas & Stern 2004). Instead of the reactive planning “active plan tree” mentioned above, ABL agents have an Active Behavior Tree (ABT) that is in constant flux during the agent’s performance.

Why ABL?

ABL will be the agent architecture on which we explore authoring support for this dissertation. ABL, like its predecessor Hap, was designed for the believable, dynamic agents previously described, and is robust enough to support a publicly releasable interactive experience. *Façade* via its existence proves that ABL is complex enough to support high-quality agents with robust behaviors, while also managing to scale at least semi-tractable and performing without crashes or developer hand-holding on audience machines. Because of its potential agent complexity and full feature set, ABL is also ripe for authorial assistance for anyone who is not currently an expert with the architecture.

Use Cases

Mateas and Stern were able to leverage the dynamic, reactive behavior tree at the core of ABL to direct the agents Trip and Grace through complex and dramatically rich interactions with a first-person player and between each other (Mateas 2002). Now, over ten years later, *Façade* is cited as one of the only fully realized dramatic experiences created making use of these behavior tree research advances. However, other uses of ABL agents include procedural level generation in Launchpad (Smith et. al. 2011) and Tanagra (Smith, Whitehead, & Mateas 2010), StarCraft (Blizzard Entertainment 1998) agents (McCoy & Mateas 2008; Weber, Mateas, & Jhala 2010), and full body agent interaction in IMMERSE (Shapiro et. al. 2013).

Idioms

Weber et. al. published an overview of the design patterns for EISBot, their StarCraft (Blizzard Entertainment 1998) player, as general reactive planning idioms (Weber et. al. 2010). While for a different domain than dramatic characters, the idioms in the paper serve as common tasks that different authors use when authoring in ABL. I would like to use this as a starting point for the idioms to be reified in my proposed work, so I will enumerate the idioms with brief summaries and examples here:

Daemon Behaviors enable agent multitasking via a listening behavior with its ear constantly to the ground, ready to pursue any goals sent its way by using the `Spawngoal` keyword.

Messaging is the idiom for sending information between different parts of the ABT through WMEs. One behavior waits for the existence of a message, and another behavior creates those messages. The first behavior can then consume the message by pulling relevant information out of it and then deleting the WME.

Managers is the general term for partitioning parts of the ABT for different tasks. Any ABL program of any respectable size creates their own system of managers to organize their code, but there is no consistency in this manager organization between ABL programs or programmers. One of the major aspects of this document's proposed work is to define a set of managers for creating ABL character agents that is then reified in the authoring tool architecture.

Micromanagement Behaviors are the lower-level behavior companions to the high-level *managers*. These behaviors enact all the final decisions of the agent once the decision-making reasoning has already been done elsewhere.

Mateas and Stern have described their ABL joint action idioms used in *Façade*, which are much more multi-agent focused, involving agents that have independent ABTs (Mateas & Stern 2004). These joint idioms primarily involve negotiating joint behavior participation and handling conflicts. Other idioms include player input handling behaviors, higher-level beat behavior organization, cross-beat behaviors, and body resource management (Mateas & Stern 2004; Mateas 2002).

Dan Shapiro et. al., in ABL's most recent incarnation and implementation, described their Social Interaction Unit (SIU) idiom: a way of nominating behaviors by their level of importance and grouping a series of related behaviors into a few seconds of performance similar to Façade's beats (Shapiro et. al. 2013). Their performance manager handles SIU nomination and body resource management; the wrap-on mechanism flavors behaviors based on specified mood/emotional variations; and the volition process helps manage agent intention to decide which SIUs to execute. In these processes, we see the requirements of believable agents result in many complex and interlocking mechanisms that are constantly nominating and enacting different behaviors in different ways.

Agent Authoring

Authorial Burden

As far as the author is aware, there has not been a robust definition of authorial burden in the context of computer science for creating interactive digital artifacts. Rather, most users of the term mean it to refer, in general terms, to the amount of time or effort required to create some or all hand-authored content, whether the content is related to embodied agents (Mateas 2002), dialog (Reed et. al. 2011), environmental assets (Smith, Whitehead, & Mateas 2010), or many other types of design concepts in computer science. This content is everything that is *not* the architecture itself, but the data that the architecture uses in order to generate or display desired output.

Heckel et. al. conducted a study on the representational complexity of reactive agents in an attempt to compare high-level agent representational techniques and their effect on authorial burden (Heckel, Youngblood, & Ketkar 2010). The paper admits to the difficulty in comparing architectures that cannot represent the same agent or agent capabilities, but attempts to compare Finite State Machines (FSMs), Hierarchical Finite State Machines (HFSMs), and Subsumption Architectures anyway. While the paper only examines behavior trees in their most basic form (ie not reactive planning), the paper reports that reducing the number of states and transitions reduces the representational complexity, which should make authoring easier.

Authorial Leverage

Known by *authorial leverage* and sometimes *agency* or *affordances*, this concept represents the power of the author to express their will or imagination on the content they are creating. In the evaluation of authorial leverage on drama managers, Chen et. al. define authorial leverage as “the power a tool gives an author to define a quality interactive experience in line with their goals, relative to the tool’s authorial *complexity*” (emphasis from the original source) (Chen et. al. 2009). We will describe authorial complexity shortly, but for the time being we can examine two approaches to authorial leverage. One approach is described by Mateas in his dissertation: “the authorial affordances of an AI architecture are the ‘hooks’ that an architecture provides for an artist to inscribe their authorial intent in the machine” (Mateas 2002). In other words, a feature set of levers, knobs, and other controllers to direct the architecture: creative potential and the freedom to express it.

The other approach is closer to Chen et. al.’s use of leverage in the rest of their paper: that leverage can be influenced by the complexity. If the content they are authoring becomes impenetrably complicated, the author no longer feels free to make changes because of the high cost of editing previous content. While the tools of the architecture have not changed from the beginning of the project to the end, the author feels chained by the weight of previous decisions. This is an important aspect of authorial leverage to acknowledge and consider for our proposal, as agent authoring is extremely complex and threatens to reduce authorial leverage very quickly.

Authorial Complexity

In the introduction, we introduced the concept of authorial complexity: a different approach to examining authoring burden, which is very close to Heckel et. al.'s representational complexity. However, while counting states and connections is very quantifiable, it fails to capture the challenge of authoring the states such that they are *believable*. There is an undeniably human element in behavior authoring where the author must imagine (or see through debugging processes) the agent in their state enacting each connection: does it look natural? How would the agent handle this particular transition? Would every agent handle it that way? Ensuring (or debugging) believability in this painstaking manner is required to create believable agents. *Authorial complexity* refers to the complexity of authoring a given piece of content (in this case, a behavior), which involves sifting through and making use of authorial leverage. This is why Chen et. al. says leverage is relative to complexity: authors cannot make use of the leverage that they do not understand (Chen et. al. 2009). A summary of all these elements can be seen in Figure 3.

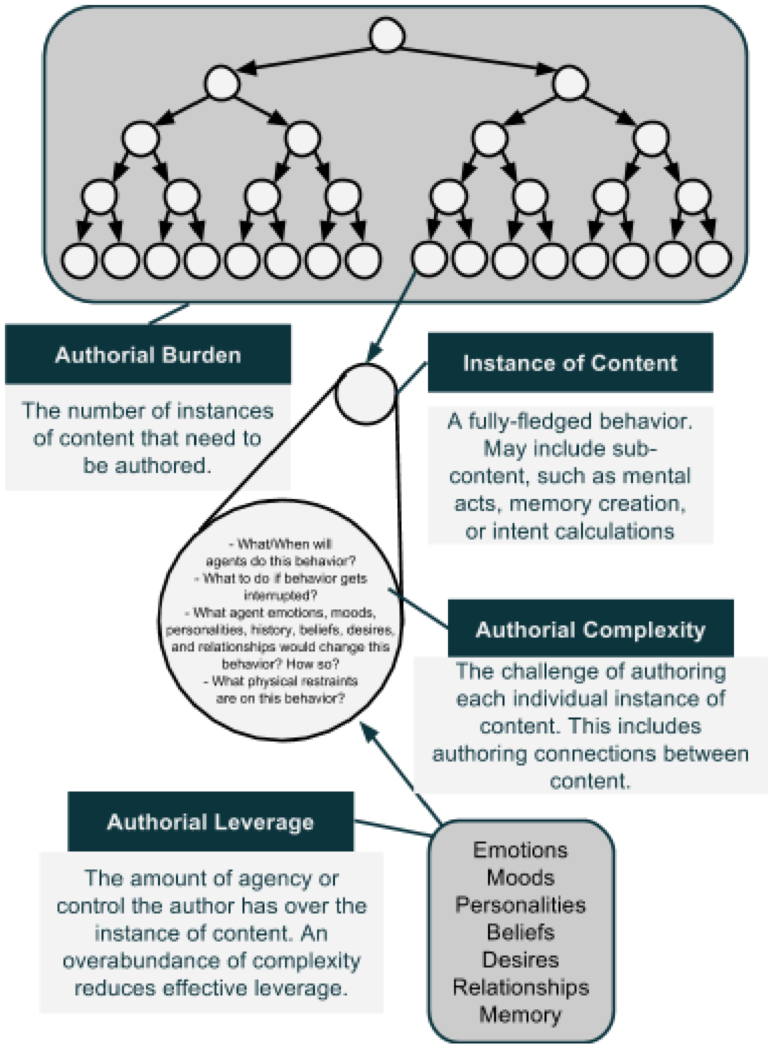

Authorial Burden
The number of instances of content that need to be authored.
Instance of Content
A fully-fledged behavior. May include sub-content, such as mental acts, memory creation, or intent calculations
- What/When will agents do this behavior?
- What to do if behavior gets interrupted?
- What agent emotions, moods, personalities, history, beliefs, desires, and relationships would change this behavior? How so?
- What physical restraints are on this behavior?
Authorial Complexity
The challenge of authoring each individual instance of content. This includes authoring connections between content.
Authorial Leverage
The amount of agency or control the author has over the instance of content. An overabundance of complexity reduces effective leverage.
Emotions
Moods
Personalities
Beliefs
Desires
Relationships
Memory

Figure 3: A summary of authorial burden, complexity, and leverage in the context of believable agents represented via a behavior tree. The tree at the top represents the final scenario. Each bubble is a behavior, which incorporates all desired elements of believability, and interfaces with other behaviors in the tree.

While the ABL architecture has been designed to make simple character behaviors need only a few lines of code (Mateas & Stern 2004), designing *robust behaviors*, behaviors that can interact appropriately and believably with many other behaviors, is extremely challenging. That is, ABL offers a huge level of leverage, so much so that trying to make use of it all blows authorial complexity out of proportion. Other researchers have noted this problem:

> *The cost of reactive intelligence is engineering. The agent's intelligence must be designed by hand since it performs no search (including learning or planning) itself… Although behavior modules are themselves definitionally easy to create, engineering interactions* between *behaviors has proved difficult.* (Bryson 2003)

Isla echos this observation when he says, "Quantity, of course, *is* complexity," although his term for authorial complexity is "mental bandwidth" (Isla 2005). He describes the loss of what we've been calling authorial leverage: "when we lose the ability to reason about what's going on in the system, we lose control over it." Additional context and details for these definitions can be found in the proposed work.

Authorial Evaluations

Internally, authoring challenges and tools get a wide variety of treatment, although it is difficult to find many publications describing this (hence, internally). Some systems create bug-ridden authoring tools for producing content, such as the Comme il Faut's authoring tool for Prom Week (McCoy et. al. 2011). The tool would periodically crash, lose authored content, and fail to display content properly to authors, but it was still used to make thousands of micro-theories for the game. FAtiMA only describes its authoring process: "FAtiMA characters are complex to author since their behaviours emerge from a combination of multiple factors… Since there are no conventional definitions of such values, authors have to adjust them through trial and error" (Berardini & Porayska-Pomsta 2013). In the previous work section, we show some more detailed accounts of the FAtiMA authoring process. CADIA's authoring tool, Populus, asserts an easy of authoring without providing many details as to how: "the tool is written in Python which assures flexibility, fast prototyping and deployment of new scenarios and behaviours" (Pedica 2009). Informal discussions with other authors have admitted to keeping scenarios to a small scope to minimize authoring and remove the need for a tool (Grow et. al. 2014).

BOD/POSH has deployed authoring tools to undergraduate computer science students and graduate classes of social and behavioral sciences (Brom et. al. 2006). In order to make authoring as approachable as possible, one of their major goals was "making *simple things simple, and complex things possible*" (emphasis in the original). Their approach also supported rapid prototyping and a graphical virtual environment in order to visualize agent behaviors. Authoring and visualization modules were separated into the BOD/MASON view that showed behaviors running, and the ABODE behavior editor that created the trees. However, not much detail is given on the author's experiences using the tool. We can only extrapolate that the modular approach and focus on simplicity made authoring *possible* by these students.

Weber, Mateas, & Jhala used an unpublished ABL authoring tool to visualize ABL's Active Behavior Tree when conducting their research on their StarCraft agents (Weber, Mateas,

& Jhala 2010; Blizzard Entertainment 1998). However, Weber never reported on its use, nor provided any documentation, and future ABL authors on IMMERSE found the tool to be intimidating, buggy, and overall not worth using. However, the tool is a first attempt at providing ABL-specific authoring support. Many of the same things Weber visualized in his tool, we will be trying to visualize in a more modular and user-friendly manner.

Some architectures claim to enable non-programming experts to create content related to agents, such as scene composition in SceneMaker (Gebhard et. al. 2003), and usually are presented in some form of scripting or XML-editing interface to the architecture such as XSTEP or BML (Huang, Eliëns, & Visser 2003; Kopp et. al. 2006; Vilhjálmsson et. al. 2007). However, XML and spreadsheets are not conducive to building creative agents (O'Keefe 2010). One argument presented in that article is that XML removes any formatting (for text) or visualization of the final output from the authoring process. The XML is clearly closer to the technical representation of the content rather than the author's creative mental image, which interferes with creative expression. Other authors crave the restrictions found in richer user interfaces: for example, only allowing numbers in certain text fields, or not allowing spaces in others. Another article explains that one of the reasons XML is seen as so troublesome is that it's trying to bridge a very large gap between machines and human-readable content (Bone 2002).

Although agent architectures are often lacking in their authoring descriptions, there are commercial behavior tree authoring tools used by the industry. Behave, a library for Unity, is a drag-and-drop interface for constructing behavior tees (Johansen 2014). However, not only are these behavior trees static (rather than dynamic), but they focus on tasks related to navigation and combat rather than rich, dramatic, and social interaction between agents. A research-focused behavior tree editor, AIPaint, also builds behavior trees for space-oriented behaviors for pathfinding-like tasks, this time with a sketching interface (Becroft et. al. 2011). In summary: complex dramatic agents with many interactions between their behaviors are very difficult to control and visualize. Authoring interfaces for architectures that handle dramatic behaviors are either nonexistent, poorly represented, or lack author-focused evaluations. These poor representations have led to the general, yet structured, authoring proposals in this document being defined as research contributions.

Evaluation Model for Leverage

Chen et. al. conducted an evaluation of authorial leverage for drama managers by comparing the output complexity and quality of authored stories from a drama manager and a script-and-trigger approach (Chen et. al. 2009). They showed that a drama manager provides authorial leverage, which was defined above. While the study examined stories (not agents) and decision trees (not behavior trees, nor dynamic), it is the most detailed author-focused evaluation of AI architectures we have seen. The metrics they examined were complexity, easy of policy change, and variability of experiences:

Complexity: Examining the smallest tree that achieves reasonable performance and qualitatively examining whether it would be reasonable to hand-author. Or, approaching from the other direction, beginning with a small world and measuring how complexity grows when adding additional elements. (Quantity)

Ease of policy change: Assuming a fully realized tree, how difficult is it to tune and alter the experience? Script-and-trigger alterations require many complicated changes throughout the system, while the drama manager offers high-level controls. (Quality)

Variability of experiences: Examines the variety of experiences by measuring the frequency of variability. Attempts to capture a breadth of different high-quality experiences (Quantity & Quality)

All of these metrics attempt to capture elements of decision tree authoring that we wish to also examine in behavior tree authoring. The Evaluation section near the end of this document proposes our application of these metrics to evaluate our authoring interface as Chen et. al. evaluated their drama manager.

Previous Work

The author has had a long infatuation with enabling the creation of various *things* by other authors, including arts and crafts, natural language, and embodied agents. The following research is constrained to be at least semi-related to the field of embodied agents research.

Disembodied Agents

While not directly fitting with the full form of embodied agents we've described, the author was involved with natural language generation and behavior authoring for chatbot-like agents (with at most static visual image representations).

SpyFeet

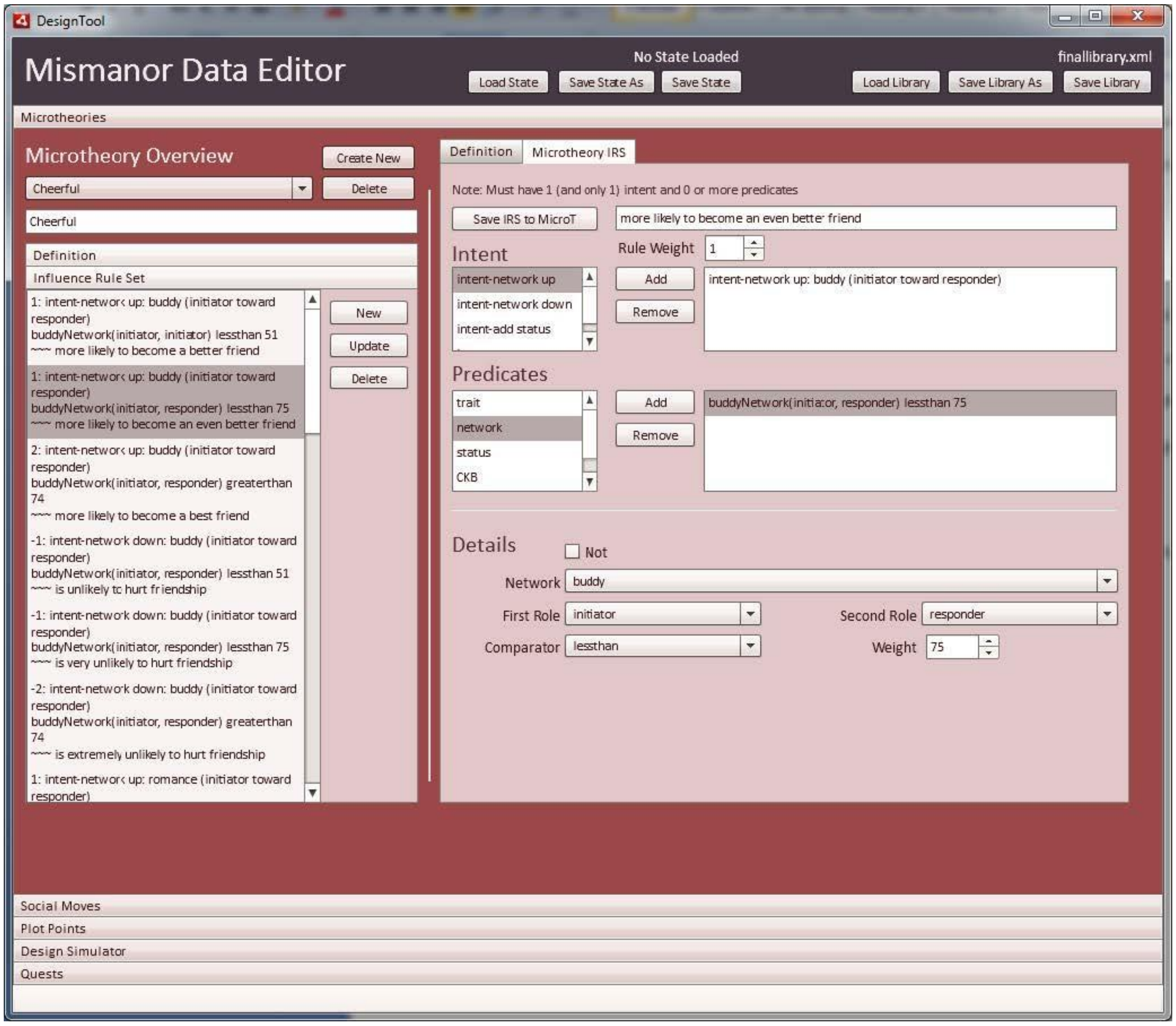
SpyFeet was a mobile app developed to motivate young girls to exercise, and generating dialog for its characters was the first time the authored encountered friction with novice authors (Reed et. al. 2011). The novice authors tasked with writing agent dialog had difficulty working in spreadsheets and situating their mental model of characters in arbitrary situations. This is a similar problem to imagining agent state space and the combinatorial number of possibilities to account for when authoring agent behaviors.

GrailGM

The author encountered further challenges from novice authors in helping Anne Sullivan with her dissertation work on quests for social role-playing games (Sullivan 2012).

> *The Grail Framework in its entirety is designed to create a framework in which the designer is able to author high-level rules together with relatively-atomic pieces of traditional content, allowing for a large amount of dynamism in play. By not requiring the designers to use traditional scripting methods to create the game story, they are able to focus on the tasks of designing and writing, as opposed to programming and scripting.* (Sullivan 2012)

While story-focused rather than agent-focused, we created a Social Mechanics Design Tool (SMDT) for Grail Framework's disembodied characters (See Figure 4).



Cheerful

Cheerful

1: intent-network up: buddy (initiator toward responder)
buddyNetwork(initiator, initiator) lessthan 51
~~~ more likely to become a better friend

1: intent-network up: buddy (initiator toward responder)
buddyNetwork(initiator, responder) lessthan 75
~~~ more likely to become an even better friend

2: intent-network up: buddy (initiator toward responder)
buddyNetwork(initiator, responder) greaterthan 74
~~~ more likely to become a best friend

-1: intent-network down: buddy (initiator toward responder)
buddyNetwork(initiator, responder) lessthan 51
~~~ is unlikely to hurt friendship

-1: intent-network down: buddy (initiator toward responder)
buddyNetwork(initiator, responder) lessthan 75
~~~ is very unlikely to hurt friendship

-2: intent-network down: buddy (initiator toward responder)
buddyNetwork(initiator, responder) greaterthan 74
~~~ is extremely unlikely to hurt friendship

1: intent-network up: romance (initiator toward responder)

Note: Must have 1 (and only 1) intent and 0 or more predicates

more likely to become an even better friend

Rule Weight: 1

intent-network up
intent-network down
intent-add status

intent-network up: buddy (initiator toward responder)

trait
network
status
CKB

buddyNetwork(initiator, responder) lessthan 75

Not: ☐

Network: buddy

First Role: initiator

Second Role: responder

Comparator: lessthan

Weight: 75

Figure 4: The Social Mechanics Design Tool for the Grail Framework. The whole tool interfaces to XML files and represents different areas of authoring in an accordion-like menu interface. Some examples of these authoring areas include Micro-theories, Plot Points, Social Moves, and Quests. The use of bullet points and drop downs restrict data entry options (and thus mistakes) where ever possible by the authors. Courtesy of (Sullivan 2012).

This design tool handled content creation for the Grail Framework at all levels of story granularity, including agent dialog, their motivations, major plot points, and whole quests. It was used by writers with varying levels of computer science education, and while all authors were able to understand the predicate logic eventually, it was a challenge for non-programmers initially. The following are lessons we learned for future iterations of the tool, many of which can be extrapolated to be used with the tool presented in this dissertation. More details about these lessons in relation to the Grail Framework can be found in Anne Sullivan's Dissertation (Sullivan 2012).

Hierarchical Confusion

The design tool was separated into sections based on the type of data being authored (See Figure 4 above). However, the authors got confused handling all levels of hierarchical data in a flat manner. For example, there were (informally) world-level, game-level, quest-level, and character-level plot points, but they all co-existed simultaneously in the same plot point editor. Character-level plot points would sometimes be coded in quest-level plot points, and it would be difficult to untangle quest and character progression later.

ABL faces a similar challenge in that behaviors are the fundamental building block in all levels of its potential hierarchical planning. It is completely up to the user's design to delineate levels of hierarchy and enforce their own consistency. A tool can help separate these hierarchical levels and help authors enter the proper design space for each level.

Context Confusion

Predicate logic was pervasive throughout all parts of the tool. The same predicate with very little visual difference could be seen acting as a precondition for a whole behavior to fire (1), as a weight to determine if an agent desired to do the behavior (2), the condition for a specific instance of the behavior (3), and the resulting value change if the behavior occurs (4):

(1) buddyNetwork(initiator, responder) greaterThan 50 (precondition)
(2) 3 buddyNetwork(initiator, responder) greaterThan 50 (influence rule weight)
(3) buddyNetwork(initiator,responder) greaterThan 50 (condition of an effect)
(4) buddyNetwork(initiator, responder) +10 (change of the effect)

(Sullivan 2012)

ABL has more explicit keywords to show when these different predicate uses are occurring (such as precondition and context_condition). However, it is important that the different functions of these predicates are clearly maintained in the tool in order to reduce user confusion.

Expected Tool Performance

The history of tool usage for all our authors brought certain expectations to their use of our tool, and we were not able to fulfill them all. Flawless saving and loading was absolutely required, and periodic auto-saving (and thus work recovery) was expected and fulfilled. However, we did not get around to implementing "undo" functionality, nor any level of copy-paste operations. Since many low-level concepts were reused when authoring similar behaviors (see those predicates above), being able to copy-paste their settings would have saved authors 3-4 mouse clicks per entry of that predicate. The authors also desired ways to filter previously authored work, such as finding all micro-theories involving a change in the "buddyNetwork" above. These are common features to data-driven tools that the ABL authoring tool should also strive to support.

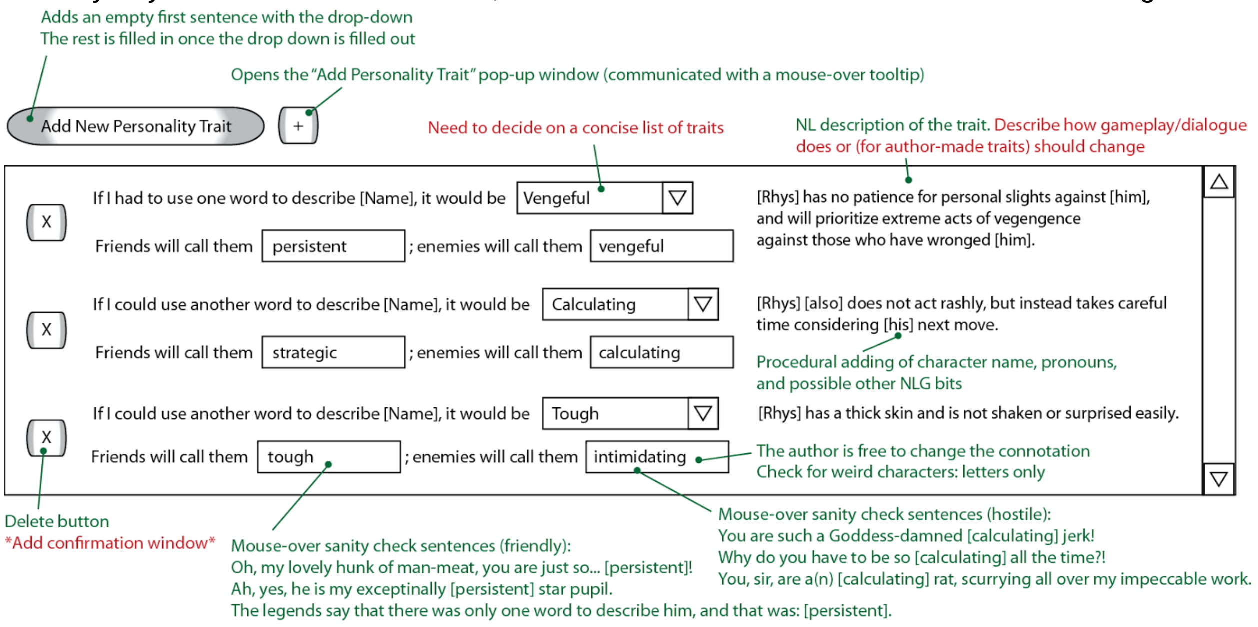
In-Tool Testing

The SMDT was a separate entity from the game that used the data that resulted from it: text files were saved and shared between the tool and the game in two separate applications. There was no verification or testing of the authored text other than those ensured by the graphical user interface (the author could not misspell an entry in a drop-down list, for example). The authors desired ways to see metrics such as reachability of certain behaviors, which predicates were constraining behavior choice, or how often micro-theory rules were used, which the tool was incapable of displaying.

Since we did not have any support in the SMDT, it is difficult to say which features would have actually been useful in aiding designers. However, The ability to compile and run ABL code within its tool is going to be critical for providing designer feedback as quickly as possible, regardless of what precisely that feedback is.

Character Creator

After GrailGM, the author built a currently unpublished authoring tool for merging the agents in Sullivan's system with the natural language of SpyFeet. This tool was designed to fill in the authoring that had not been covered in the Grail Framework, but was still required by the Framework: the agents and objects within the world, including their properties, personalities, relationships, and history. The author approached the task with the metaphor of creating a *Dungeons & Dragons* character sheet: a common trope found within many character creation systems in computer role-playing games (TSR 1977). While the tool was never finished nor used by anyone other than the author, it exists as another instance of concrete tool design.


Adds an empty first sentence with the drop-down
The rest is filled in once the drop down is filled out

Opens the "Add Personality Trait" pop-up window (communicated with a mouse-over tooltip)

Add New Personality Trait

Need to decide on a concise list of traits

NL description of the trait. Describe how gameplay/dialogue does or (for author-made traits) should change

If I had to use one word to describe [Name], it would be Vengeful
Friends will call them persistent ; enemies will call them vengeful

[Rhys] has no patience for personal slights against [him], and will prioritize extreme acts of vegengence against those who have wronged [him].

If I could use another word to describe [Name], it would be Calculating
Friends will call them strategic ; enemies will call them calculating

[Rhys] [also] does not act rashly, but instead takes careful time considering [his] next move.

Procedural adding of character name, pronouns, and possible other NLG bits

If I could use another word to describe [Name], it would be Tough
Friends will call them tough ; enemies will call them intimidating

[Rhys] has a thick skin and is not shaken or surprised easily.

The author is free to change the connotation
Check for weird characters: letters only

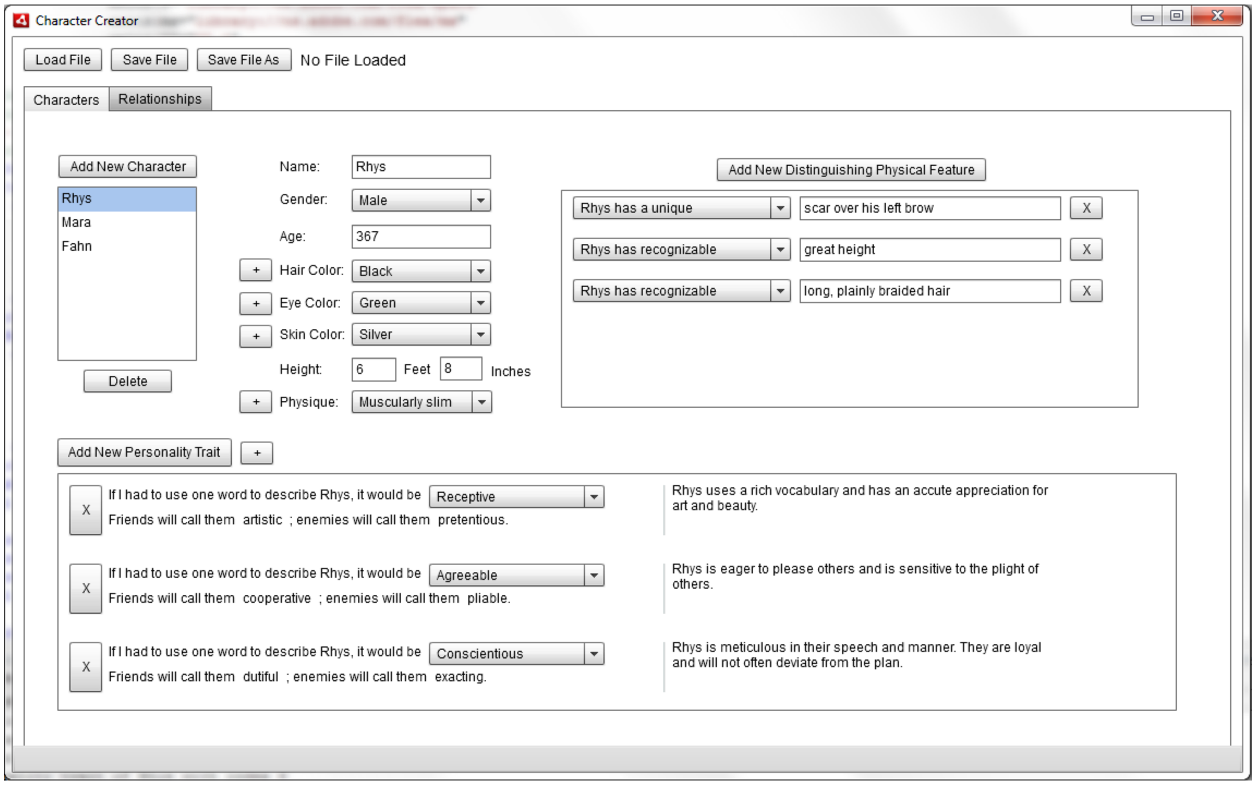
Delete button
Add confirmation window

Mouse-over sanity check sentences (friendly):
Oh, my lovely hunk of man-meat, you are just so... [persistent]!
Ah, yes, he is my exceptinally [persistent] star pupil.
The legends say that there was only one word to describe him, and that was: [persistent].

Mouse-over sanity check sentences (hostile):
You are such a Goddess-damned [calculating] jerk!
Why do you have to be so [calculating] all the time?!
You, sir, are a(n) [calculating] rat, scurrying all over my impeccable work.

Figure 5: A fraction of the Character Creator tool design spec. Where ever possible drop-down menus and radial buttons were used to verify user input. Fields were meant to update in real time to any change of variables.

Every button, text field, dropdown, checkbox, mouseover, and other user interface element was defined in a design document (see Figure 5 above). Approximately one-third of the tool was encoded and functioning (see Figure 6 below). The author had followed the design principles learned from the Grail Framework: restricted user input to ensure valid output, robust saving/loading, and simulations of how the characters and items would be used within the game.



No File Loaded

Rhys
Mara
Fahn

Name: Rhys
Gender: Male
Age: 367
Hair Color: Black
Eye Color: Green
Skin Color: Silver
Height: 6 Feet 8 Inches
Physique: Muscularly slim

Rhys has a unique	scar over his left brow
Rhys has recognizable	great height
Rhys has recognizable	long, plainly braided hair

If I had to use one word to describe Rhys, it would be Receptive
Friends will call them artistic ; enemies will call them pretentious.
Rhys uses a rich vocabulary and has an accute appreciation for art and beauty.

If I had to use one word to describe Rhys, it would be Agreeable
Friends will call them cooperative ; enemies will call them pliable.
Rhys is eager to please others and is sensitive to the plight of others.

If I had to use one word to describe Rhys, it would be Conscientious
Friends will call them dutiful ; enemies will call them exacting.
Rhys is meticulous in their speech and manner. They are loyal and will not often deviate from the plan.

Figure 6: A section of the Character Creator tool functioning. Built in Action Script, which is now considered a dead language.

Embodied Agents

While working on the next generation of Façade with Michael Mateas, Andrew Stern, and others, the author graduated from novice to tentative expert in ABL authoring. We began with Mateas' dissertation on ABL, the de facto document of ABL's most crucial elements, and made small additions to the IMMERSE project code-base with processes such as organizing a group of agents to pantomime a small group discussion. It took months of PhD-level student man-hours to get up to speed in ABL authoring to the point of working on semi-believable agents.

Over the next year, our group added structure to the code base in an attempt to both lower the authorial burden and increase or agent's believability; the SIU, performance manager, volition process, and wrap-on mechanism mentioned above were all developed during this time (Shapiro et. al. 2013).

The Social Interaction Unit

Social Interaction Units are akin to beats in Façade (Mateas & Stern 2005) and social exchanges in Prom Week (McCoy et. al. 2011). An SIU is designed to be as small as possible to accomplish a single, discrete social goal, such as a reacting to the player's approach, exchanging an object, or momentarily discussing a question. It typically lasts from 5 to 10 seconds. An SIU has a primary initiator and one or more responders, one of whom may be a primary responder. For example, the Greet SIU has a greeter (initiator), and one or more characters being greeted (responders). SIUs may take parameters, such as a character to add or remove from a group formation, a gesture to mirror, or a question being asked. SIUs can be built to allow either the player or an NPC to be the initiator or a responder.

SIUs are typically launched from higher-level decision-making rules in reaction to interpretations of social signals from another character (typically the player). Additionally, SIUs may be autonomously launched as self-motivated behavior, e.g., for maintaining a group formation. Each NPC is typically involved in multiple simultaneous active SIUs at the same time. For example, ManageFormation, ChitChat, StudyOthersObject, and ReactToQuestion may all be active simultaneously.

After the SIU pattern was established, code templates were created to ease their authoring process. An author looking to create a new SIU would copy-paste a set of partially-constructed skeleton behaviors that all shared a structure of initialization, role-selection, NPC/Player performance differences, and clean-up. The behavior templates came with connections with the performance manager so that the SIU could be used by the performance manager without the author having to worry about interfacing with those different aspects of the code.

The Performance Manager

The performance manager interleaves SIU execution using an authored-defined priority assigned to each SIU. Priority tiers are heuristically tuned to favor more important performance requests such as reacting quickly to a gesture from the player, over less urgent performance such as maintaining group formation or chitchatting. If the Performance Manager suspends a less important ongoing performance to allow a more important request to perform immediately, the suspended performance will typically re-run when the interrupting performance has completed. Since each performance request typically lasts no more than a few seconds, re-running interrupted performances often maintains coherence and believability.

Volition Process

The volition process performs social reasoning by deciding which SIUs to execute and when. The mechanism employs three types of production rules, added to ABL for this purpose. These rules leverage the existing pattern matching capabilities of ABL, but provide new functionality for asserting and processing new working memory elements. The first are nomination rules. These rules assert an SIU decision object into working memory when its preconditions hold in the social context. Nomination rules execute whenever a new social interpretation is introduced to the simulator through the input channel, or a specified amount of time elapses. This is the first step of the volition process. At this stage, the resulting SIU decision objects only indicate that the associated behaviors are possible given the social context, but they are not immediately executed.

The second type of rule integrated into ABL represent desirability, or influence. A given influence rule matches an SIU decision object plus additional state data. The action side of the rule increments or decrements a preference value for the execution of that SIU. A single aggregated score is computed for each SIU decision object from all matching influence rules, and any SIUs above a static threshold are marked for execution.

The third type of rule matches a set of relations on the left hand side and either asserts or deletes any number of standard ABL working memory elements on the right hand side. This is done for all matching variable bindings. These rules generalize the trigger functionality in CiF and are fired at a specific moment in the decision process. A separate parallel process continuously monitors SIU objects marked for execution, and launches the corresponding behavior by adding it to the active behavior tree (ABT).

Wrap-On Mechanism

The wrap-on mechanism addresses the problem of encoding character performance given a wide variety of behaviors and attitudes. A naïve approach to capturing these variations quickly leads to an intractable amount of authoring, so we focus, instead, on building reusable social affordances that can be authored independently and recombined dynamically to produce the desired performance range. The wrap-on mechanism does this by layering expressions of attitude onto behavior.

In more detail, we author the core functionality of each SIU in an affect-neutral way, comprised of very short functional performances. For example, ExchangeObject has the initiator holding out the object, waiting for the responder to reach for it, which finally completes the exchange. Next, we write behaviors that can perform these low-level actions with a variety of attitudes. This can be as simple as defining attitudinal variations of individual animations, such as holding out an object in a friendly, aggressive or hesitant way. The wrap-on manager acts by inserting these animations before, after, and/or in place of the core performance, such as impatient sighs, frustrated gestures, or nervous glances, using the meta-behavior capabilities in ABL to alter the structure of the ABT.

Requirements Analysis for ABL authoring

In an effort to approach creating authoring support for ABL, we conducted a survey of embodied agent authoring architectures and uncovered interesting similarities in how reasonably isolated groups each approached their authoring challenges (Grow et. al. 2014). In preliminary conversations with five different research institutions, we found that many of them do not believe the cost of an authoring tool is less than the effort to brute-force simple agents to prove their research goals. However, authoring tools would be needed to scale up production of their agents because of the rapidly increasing authorial complexity. We set out to find out what the authoring process was for these systems so that we could begin to understand how to approach reducing their authorial burden and complexity.

The System-Specific Step

At a high-level, we carved out a chunk of the authoring process, which we called the System-Specific Step (SSS), as the term for the parts of the authoring process where broad discussion ends and system-specific design constructs are used instead. The SSS is a troublesome idea because it both represents a phase in the authoring process that all IVA authors must complete, but how each author approaches, conceptualizes, and executes the phase can be completely different. We set out to map exactly how each author approached, conceptualized, and executed this phase with as much detail as possible for scientific examination.

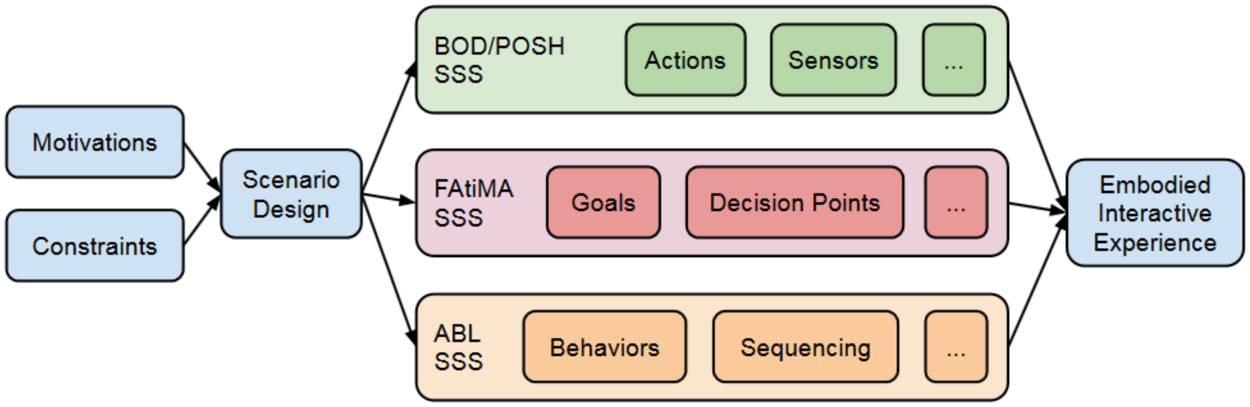


Figure 7: Illustrates the boundaries of the SSS for the three different agent architectures that we conducted case studies with.

In order to find a specific system's point of divergence from shared authoring concepts, we needed to run the architecture through an authoring exercise. We designed a simple scenario, described below, for intermediate-expert authors to transform into descriptive pseudocode for their own system, one step removed from actually programming the scenario. These intermediate-expert authors were accompanied by an analyst who was not an expert in the architecture, which forced the intermediate-expert to elaborate and make explicit every step of their authoring process.

The authoring team then translated their work into a rigorous process map, where the analyst wrote the map (confirming their understanding) and the intermediate-expert validated and expanded it as necessary. Process mapping involves creating a visual representation of a procedure similar to a flow chart, making explicit "the means by which work gets done" (Madison 2005). Details of each step (and possible sub-steps) in the process were recorded, such as the duration of each step, other people involved, and possible authoring bottlenecks. The goal of this process map is to make as much of the authoring process as explicit as possible for analysis.

In the following case studies, conclusions drawn from the process maps of each team are enumerated as SSS Components. We found this process not only helpful, but necessary to discover actionable means by which to improve the authoring experience for the requirements analysis. It is important to note that it may take multiple of these sessions with the same author (and possibly multiple process maps) to obtain the full authoring process with sufficient detail for analysis. For example, one early process map made with an ABL author was very high-level, focusing on the interconnection with other teams and the bottleneck this caused. The analyst had to return to the ABL author for another session aimed at creating a new process map through the expansion of a single node in the first process map. All the involved systems have been mentioned and described previously in the related works section, so their introductions are not necessary.

The Scenario

The scenario we chose is a simplified version of the "Lost Interpreter" scenario recently completed and demoed by the IMMERSE group in ABL (Shapiro et. al. 2013). It involves the player as an armed soldier in an occupied territory searching for their lost interpreter via a photograph in their possession. The player must show the image to a cooperative local civilian, who will then recognize the person in the photograph and point the player in the direction of the interpreter. Once the player knows the location, the scenario is successfully completed. The uncooperative civilian will not respond to the player's pleas for help, and if the player is rude or breaks social norms (Evans, forthcoming), the NPCs (Non-Player Characters) will leave and the scenario will end unsuccessfully.

We chose this scenario because it exercises a wide range of capabilities of interactive characters: player involvement, multiple NPCs with different attitudes, a physical object, communication between NPC and PC, and multiple outcomes. The scenario was also simple enough that each team was able to reach a pseudocode state of completion in a reasonable amount of time (1-3 hours). While the original IMMERSE scenario required non-verbal communication via gesture recognition, we did not enforce that modality on other systems. The specifications of the scenario were designed to be loose enough to allow each system to encode the scenario to their system's advantages without demanding extraneous features that all systems may not possess.

Case Study 1: BOD using POSH

A programmer and designer worked together to create a list of abstract behaviors that need to be performed. It is important to note that the BOD designer (as distinct from the programmer) will never need to encounter anything more complicated than graphical interfaces in their authoring interactions, allowing the designer and programmer to be the most independent of the three case studies (although they may be the same person in some projects). The separation of these two roles is part of the design philosophy of BOD. In our case study the abstract behavior list included seven actions: a greeting/goodbye to mark the beginning and end of the interaction, accepting, examining, returning an item, ignoring the player (for the uncooperative agent), and telling information. The second step was to build what is ultimately a list of procedure signatures for the programmer, determining which of these behavior elements need to be represented as actions and sensors, as well as an idle state should all else fail (Bryson & Stein 2001).

The programmer then coded the actions and sensors as functions to create the building blocks of the dynamic plan. In parallel, the designer used the primitives (actions and sensors) created by the programmer to design the behavior tree using ABODE*, a graphical design tool for POSH plans.

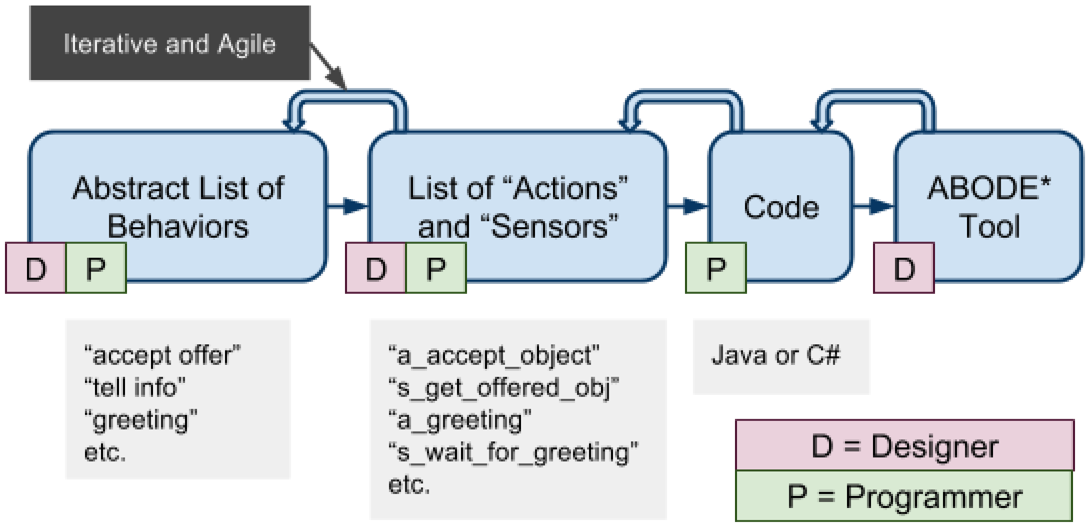


Figure 8: The BOD/POSH SSS in detail.

Case Study 2: FAtiMA

When presented with the requirements of the the Lost Interpreter scenario, FAtiMA authors started by considering the motivations of the NPCs. Since the behaviors of agents in FAtiMA are goal driven, it was proposed that NPC's in this scenario must have the explicit goal of helping the player, which we creatively called *Help*. Additionally, there needed to be a motivation not to help, in order to model the uncooperative NPC's behavior. The authors chose for the uncooperative NPC to have the goal of avoiding harm from the armed player (let it be called *ProtectSelf*). For this second goal to be useful, there must be an NPC action that is helpful to the player, but at the same time might put it in harm's way. For instance, the NPC might consider the possibility of being harmed when taking the picture from the player. If the agent considers a plan involving possibly being harmed, then it will feel a Fear emotion. The authors then continued to define actions that the agents can take along the path of reaching the help goal, such as actually taking the photo, examining it, or speaking.

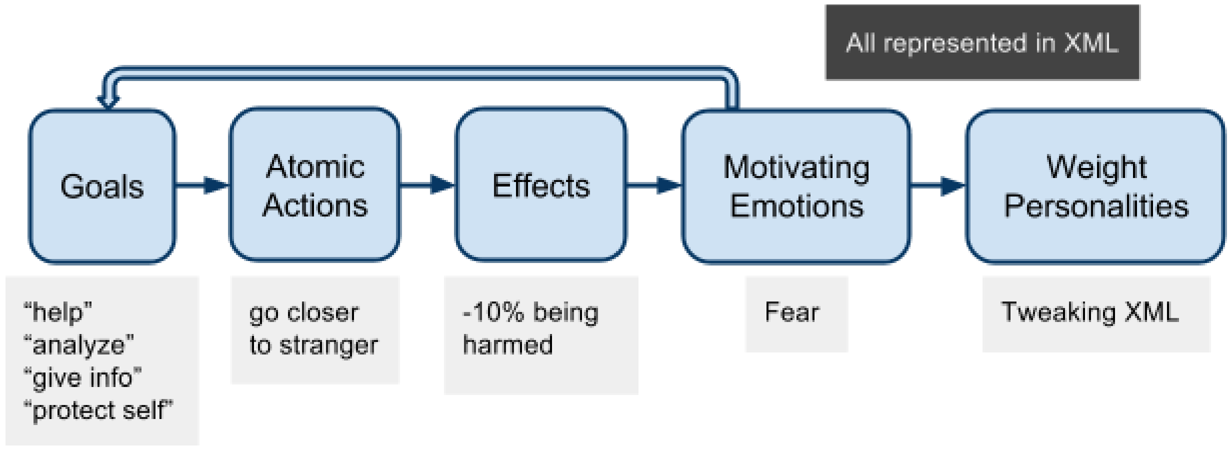
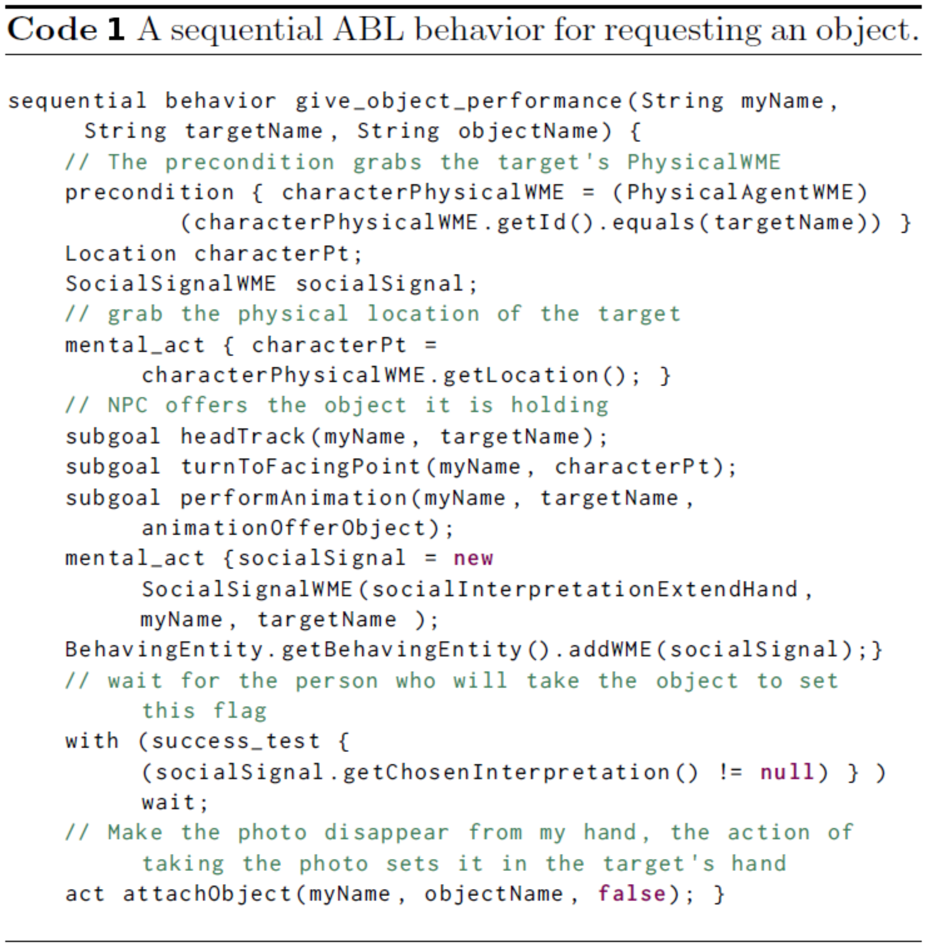


Figure 9: The FAtiMA SSS in detail.

Case Study 3: ABL

The ABL authors approached the scenario by first creating a list of abstract behaviors which were stubbed into the ABT in a rough sequential structure. At a high level, the authors each tackled a specific behavior and worked iteratively with each other to bring it to completion. Consider the example give_object() behavior for a character to hand an object to another character:



Code 1 A sequential ABL behavior for requesting an object.

```
sequential behavior give_object_performance(String myName,
    String targetName, String objectName) {
   // The precondition grabs the target's PhysicalWME
   precondition { characterPhysicalWME = (PhysicalAgentWME)
         (characterPhysicalWME.getId().equals(targetName)) }
   Location characterPt;
   SocialSignalWME socialSignal;
   // grab the physical location of the target
   mental_act { characterPt =
       characterPhysicalWME.getLocation(); }
   // NPC offers the object it is holding
   subgoal headTrack(myName, targetName);
   subgoal turnToFacingPoint(myName, characterPt);
   subgoal performAnimation(myName, targetName,
       animationOfferObject);
   mental_act {socialSignal = new
       SocialSignalWME(socialInterpretationExtendHand,
       myName, targetName );
   BehavingEntity.getBehavingEntity().addWME(socialSignal);}
   // wait for the person who will take the object to set
       this flag
   with (success_test {
       (socialSignal.getChosenInterpretation() != null) } )
       wait;
   // Make the photo disappear from my hand, the action of
       taking the photo sets it in the target's hand
   act attachObject(myName, objectName, false); }
```

Figure 10: ABL code snippet authored for giving an object to another agent.

- *The context of how the behavior will be triggered*: in this scenario, the author knows that give_object() will be triggered in response to a request_object() behavior or it will be accepted unconditionally. It contains no logic for having the offered object rejected. This behavior also only handles removing the object from the character's hand, and assumes another behavior handles the object's fate.

- *Relevant signals and WMEs*: The previous behavior was authored assuming that the characterPhysicalWME contains locational information, that there is a socialSignalWME ready to handle socialInterpretationExtendHand, and that there are constants such as the cExchangeObjectDistance previously defined and calibrated for the world. If any of these are lacking, or the author does not know about them, the author must search the existing code or create them.

- *Expected animations*: Head tracking, eye gaze, and holding out the offered object are the animations used in this behavior. The logic behind procedurally animating them is handled elsewhere, and if it were not, the author would have to create it.

- *Possible Interruptions*: The most challenging and crucial step to making these behaviors robust is handling interruptions, which the above behavior fails to do. In the success_test, if the NPC never acknowledges the socialSignal or the player never comes in range, the NPC will hold their hand out forever. If a timeout was added to holding out their hand, what should the NPC do about the unrequited object offering, and how should the lost request_object() context be handled? These are all considerations the author must address when making behaviors robust.

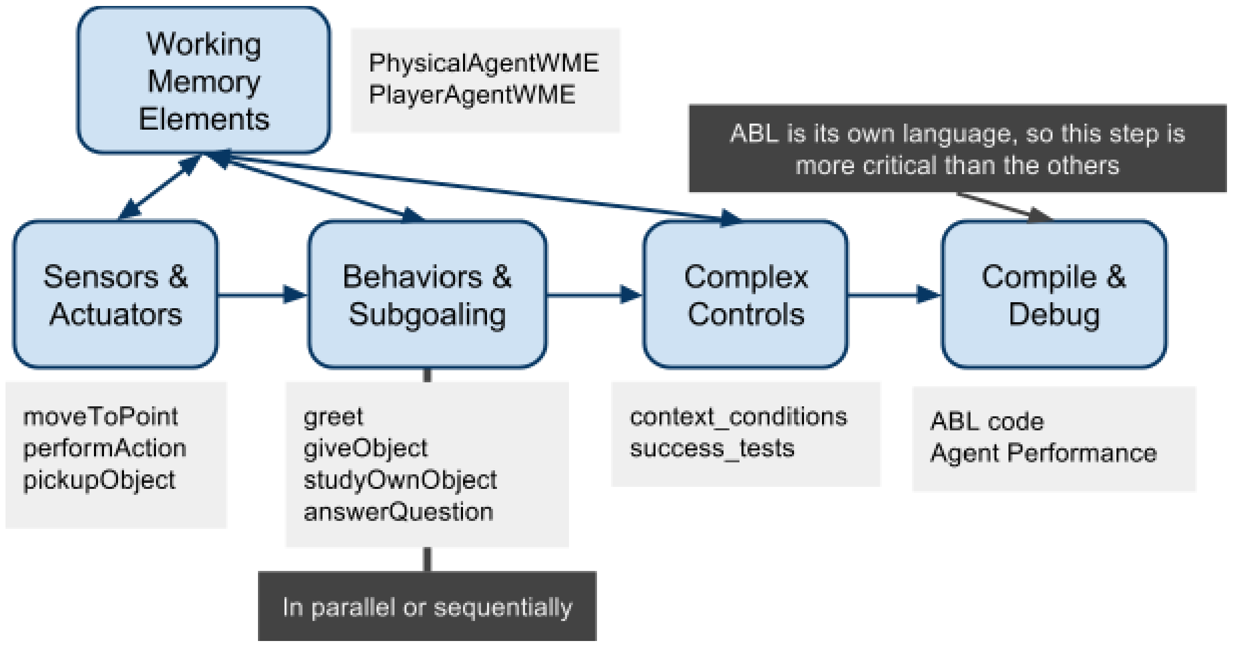


Figure 11: The ABL SSS in detail.

Discussion

In an attempt to distill the authoring process into smaller components, each team defined a set of SSS Components as major guidelines, steps, or lessons learned from the process of mapping their SSS. The summary of them is listed here, and greater detail about them can be found in the source paper (Grow et. al. 2014):

#	Name	Summary	Systems	Authoring Support
i	Start Minimally	Having a working vertical slice early gives programmers and designers a good overview of the scenario structure.	BOD/ POSH, ABL	Current ABODE* graphical design tool is sufficient.
ii	Decompose Iteratively	Filling in the stubs iteratively gives designers and programmers freedom	BOD/ POSH,	Current ABODE* graphical design tool is sufficient.

		to adjust the structure without getting in each other's way.	ABL	
iii	Minimize and Encapsulate	The BOD/POSH tree relies on simply logic to execute quickly, so complex sensory preconditions should be offloaded to behaviors.	BOD/ POSH	A module that manages encapsulated behaviors, keeping them simple and proposing them to new authors.
iv	Goals First	The agent's actions are driven by goals, so there must always be a goal structure.	FAtiMA	Combined with SSS Element v.
v	Find Decision Points	Necessary scenario-defined decision points make sub-goals more apparent to author.	FAtiMA	Scenario event sequencing tool with prompts for goals and actions at decision points.
vi	Goal Weighting and Tuning	Agent's different behaviors are driven by different weights, which is a huge time-sink to debug.	FAtiMA	Parallel execution and real-time adjustment/ comparison of values
vii	Intent Goals for Future Consequences	Language-specific limitations, such as only having one active goal at a time, hinder novice-intermediate authors.	FAtiMA	Better documentation
viii	Define Coding Idioms	As ABL is its own language, an author must have a strong understanding of their chosen idioms before coding.	ABL	Too advanced for a tool to offer much help
ix	NPC and Player Considerations	An author must conceptualize roles, the contents of the working memory and ABT, and fine-grain performance details while building up their behaviors.	ABL	Revival of the ABL Debugger through modularization: offline code analysis of behavior structures through idioms
x	Consider Interruptions	Authors must try to robustify their behaviors against interruptions and stalling, which complicates the previous SSS Element.	ABL	Revival of the ABL Debugger through modularization: tree visualization of iterations and disparate tree sections.

Table 3: The summary of SSS Elements found in (Grow et. al. 2014).

The SSS Components that arose from the simplest case study, BOD/POSH, were high-level authoring guidelines that apply to multiple architectures. Specifically, all three of BOD/POSH's Components apply to ABL as well, as they are more characteristic of a hierarchical planning structure than of BOD/POSH specifically. Other SSS Components, such as FAtiMA's Goals First (iv), are guided by FAtiMA's planning-oriented cognitive-appraisal architecture that is driven by explicit goals. The ABL case study provides a level of complexity above the other two; ABL is a general reactive planning language where many authoring idioms may be designed, as well as the only architecture in the case studies with a behavior tree that dynamically changes during runtime.

Many interviewees were resistant to the idea of specifying implementation time (in number of hours), as it varied greatly between each task. We also found that the particular shape or contents of any single process map wasn't as relevant as the process of elucidation and reflection. The goal of the process mapping technique is to tease out what is general and what is system-specific about a given architecture. The system-specific information forms the core of requirements analyses and the actionable plans found therein. The previous table shows a summary and consolidation of each team's analyst's best attempt at discovering system-specific patterns of frustration and proposing solutions to alleviate the problems.

Although the SSS concept contains the phrase "System-Specific" in its name, we found that certain SSS Components are shared between different systems, revealing common architectural tropes. However, we did find common medium-level authoring challenges that may be of use to other teams by abstracting SSS Components of the case studies: the need for (better) mechanisms for behavior (or other architecture construct) sharing and reuse, live debugging, and template structures for architecture constructs. Even though our authoring process diagrams for the System-Specific Step did not include any architectural coding examples, we did ask authors to describe their debugging process and how they would go about the coding process. We came up with a generalized and shared authoring process diagram:

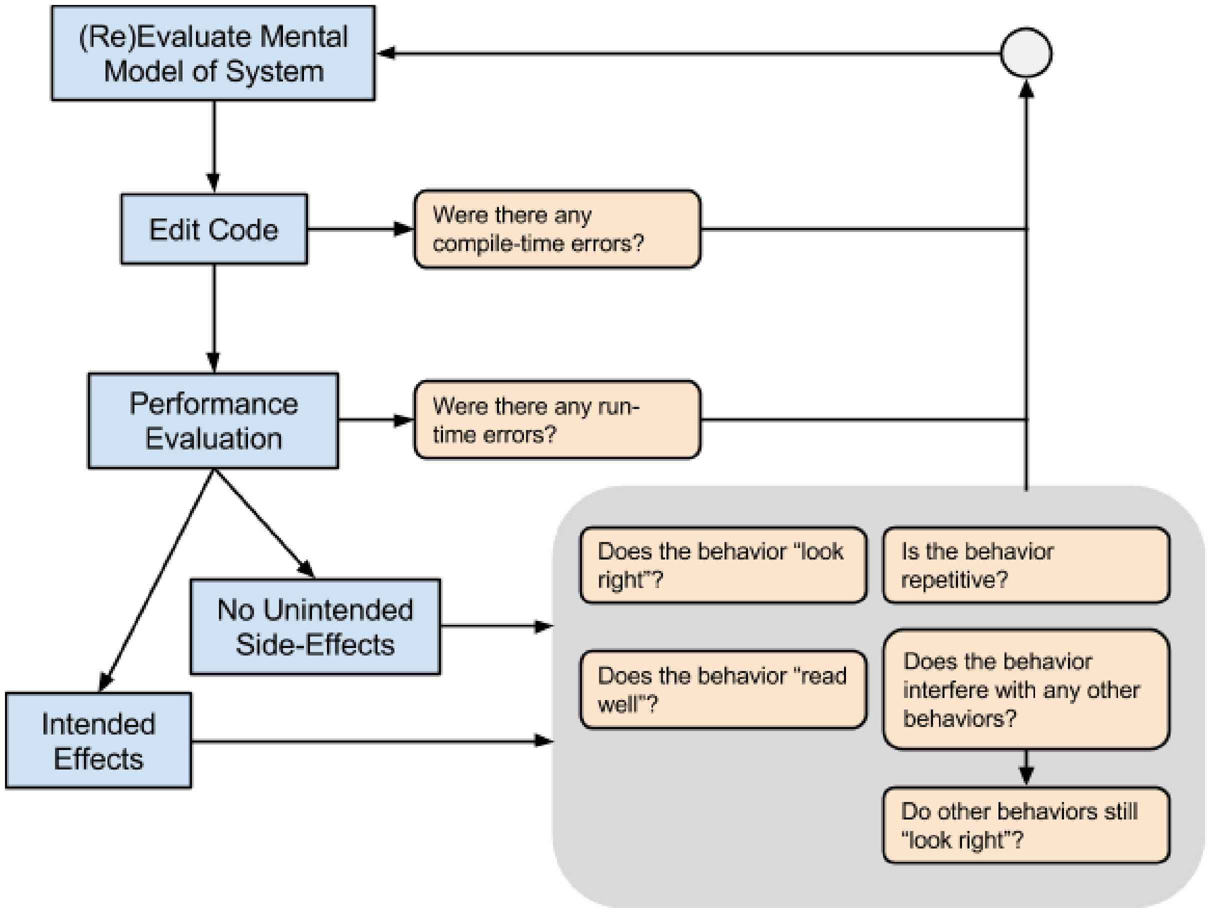

(Re)Evaluate Mental Model of System
Edit Code
Were there any compile-time errors?
Performance Evaluation
Were there any run-time errors?
No Unintended Side-Effects
Intended Effects
Does the behavior "look right"?
Is the behavior repetitive?
Does the behavior "read well"?
Does the behavior interfere with any other behaviors?
Do other behaviors still "look right"?

Figure 12: The authoring process diagram that every author across the different architectures followed.

Proposed Work

The following work proposes two primary bodies of work: a general set of authoring support strategies and patterns to be used by any agent architecture, and an application of these support strategies and patterns to the ABL architecture. The work is informed by the previous sections of related work and prior work in agent design with regards to behavior trees, authoring experience and feedback, and authoring tool design.

Authoring

Definitions

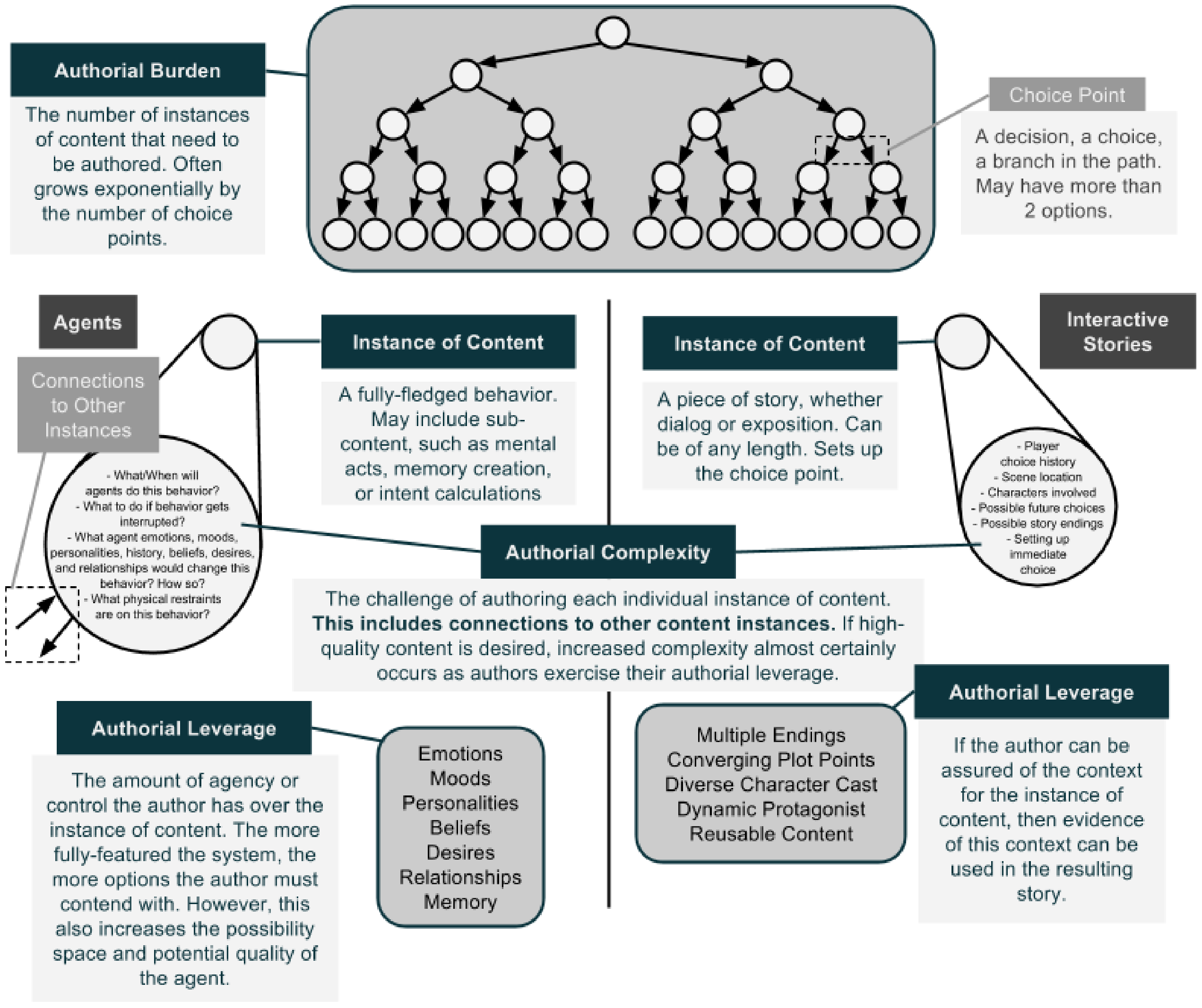

Authorial Burden
The number of instances of content that need to be authored. Often grows exponentially by the number of choice points.
Choice Point
A decision, a choice, a branch in the path. May have more than 2 options.
Agents
Connections to Other Instances
- What/When will agents do this behavior?
- What to do if behavior gets interrupted?
- What agent emotions, moods, personalities, history, beliefs, desires, and relationships would change this behavior? How so?
- What physical restraints are on this behavior?
Instance of Content
A fully-fledged behavior. May include sub-content, such as mental acts, memory creation, or intent calculations
Instance of Content
A piece of story, whether dialog or exposition. Can be of any length. Sets up the choice point.
Interactive Stories
- Player choice history
- Scene location
- Characters involved
- Possible future choices
- Possible story endings
- Setting up immediate choice
Authorial Complexity
The challenge of authoring each individual instance of content. This includes connections to other content instances. If high-quality content is desired, increased complexity almost certainly occurs as authors exercise their authorial leverage.
Authorial Leverage
The amount of agency or control the author has over the instance of content. The more fully-featured the system, the more options the author must contend with. However, this also increases the possibility space and potential quality of the agent.
Emotions
Moods
Personalities
Beliefs
Desires
Relationships
Memory
Multiple Endings
Converging Plot Points
Diverse Character Cast
Dynamic Protagonist
Reusable Content
Authorial Leverage
If the author can be assured of the context for the instance of content, then evidence of this context can be used in the resulting story.

Figure 13: An expanded Figure 3 diagram. The definitions are more detailed to support the proposed work in this section. The definitions are also applied to interactive narrative, as part of the related work supporting these terms comes from that literature.

So far, we have wrangled with the concept of *authoring*, which describes content creation for or within some architecture by human authors. This broad definition includes a variety of content and architectures so vast that it is not a useful definition on its own. When describing the number of *instances of content* that need to be authored growing exponentially (in, for example, branching narratives), the term *authorial burden* describes the number of instances. We have proposed the term *authorial complexity* to define the challenge of authoring each individual instance of the content, which varies greatly and independently from authorial burden. *Authorial leverage* is a term used to refer to the human author's agency or control over what they are creating during any given instance. Authorial leverage and authorial complexity are often positively correlated (with high leverage resulting in more complex authored instances of content), but not necessarily. Finally, in previous work, we attempted to map the *authorial process* of different architectures: a process map of detailed steps that describe what exactly occurs when the author creates an instance of content for a given architecture.

The following Figure summarizes the context and details of the above definitions (except authoring process, see previous work for agent authoring process). Much of this diagram has been implied or alluded to informally, or partially presented in the terms of a given architecture, in the related work. However, we present this diagram as the beginnings of our work to define a shared and concrete vocabulary to describe content authoring. While this document is focused on agent authoring patterns and will continue to move forward in that space specifically, the below diagram shows both agent and interactive storytelling use cases because both fields tackle similarly structured authorial burdens and authoring vocabularies.

Patterns

The previous work began to shape a loose authoring process discovery method for use by general systems, and we began to find patterns in the approach and methods of the authors. Although they were using different architectures, they are creating agents with similar connecting parts toward the same goal. At a high level, the authors must keep track of all the elements in the following diagram for the purposes of debugging behaviors when they do not perform as intended:

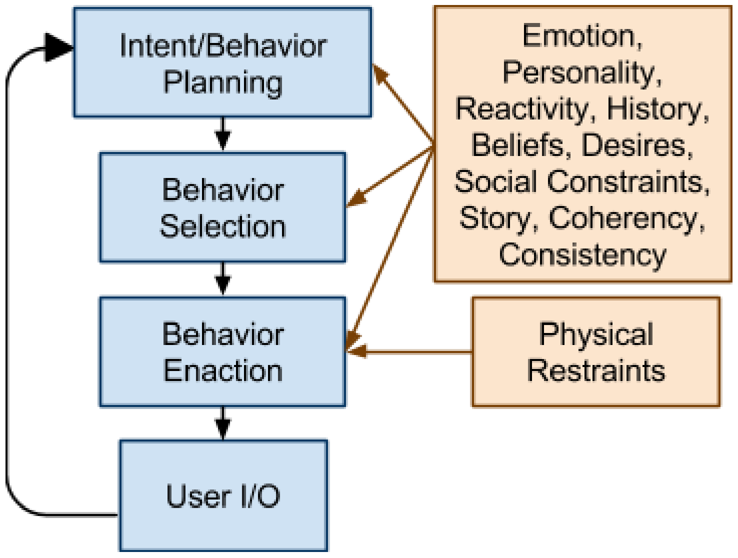
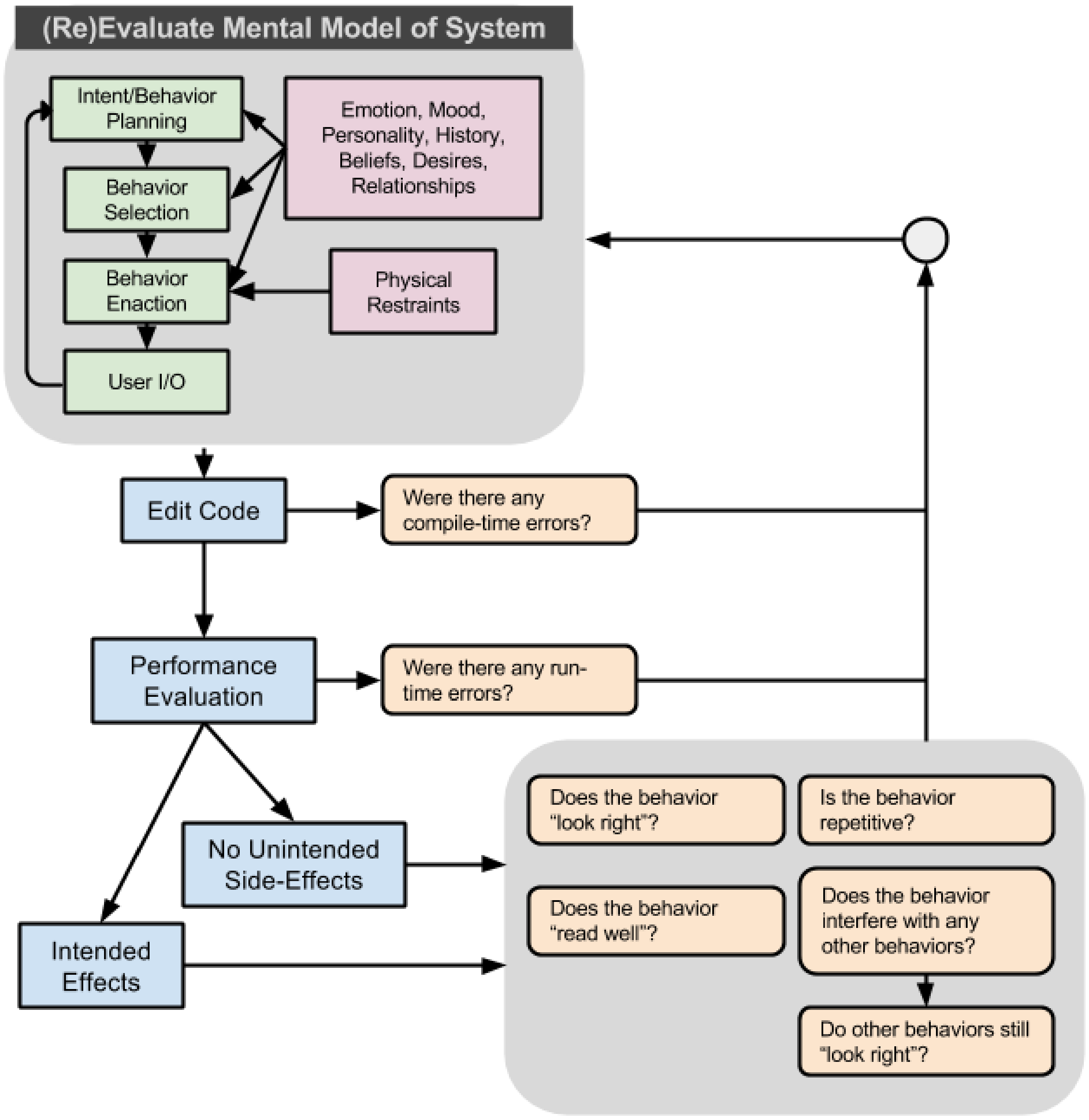

Intent/Behavior Planning
Behavior Selection
Behavior Enaction
User I/O
Emotion, Personality, Reactivity, History, Beliefs, Desires, Social Constraints, Story, Coherency, Consistency
Physical Restraints

Figure 15: A restatement of figure 1. These are elements of the architecture authors must hold in their mental model, including all subjective criteria of believable agents.

Combining this diagram with previous work, we begin to see a general description for the nebulously “difficult” authoring process of interactive dramatic agents:



(Re)Evaluate Mental Model of System
Intent/Behavior Planning
Behavior Selection
Behavior Enaction
User I/O
Emotion, Mood, Personality, History, Beliefs, Desires, Relationships
Physical Restraints
Edit Code
Were there any compile-time errors?
Performance Evaluation
Were there any run-time errors?
No Unintended Side-Effects
Intended Effects
Does the behavior “look right”?
Is the behavior repetitive?
Does the behavior “read well”?
Does the behavior interfere with any other behaviors?
Do other behaviors still “look right”?

Figure 16: Includes both the "mental model" (re)evaluation section as well as the general authoring process diagram discovered in previous work. This is why agent authoring is so hard. Now that we've aggregated them on a big map, we have to break them down into solvable sub problems.

If the author does not get their behavior correct and satisfactory on the first pass, which they almost never do, it is difficult to pin-point where corrections should be made in the code. First, there are the technical requirements of the architecture, such as a language specification, which must be obeyed for the agent and program to be compiled. Next, when the agent and scenario is run, the behavior must be triggered and performed, which may require specific scenario set-up or an introductory sequence of actions before the author reaches the proper setting for the behavior. The author must then check technical aspects of the behavior: does it trigger and finish properly, are its parameters correct, and does it change agents' state or perform its system-level functions correctly? At the same time, the author judges its aesthetic performance in regards to being believable and readable for the future audience, taking note of timing, duration, and physical asset blending. If at any time these requirements fail, the author must go back to the code to tweak numerical values, re-examine this behavior or those interacting with it for updates, or perhaps redesign the behavior from scratch if some misunderstanding has caused an unrecoverable catastrophic failure. Depending on the architecture, the author may have to request external assets, such as fresh animations, if the performance is unsatisfactory. And, of course, if any other behaviors get changed, there may be a domino effect with testing that new behavior or other behaviors reliant on those changed behaviors.

Visualization Modules

From our discussions with agent authors in previous work, we know at least a few areas where authors have had problems debugging agent behavior: reliably triggering a behavior and circumstances for consistent testing, tweaking performance metrics until they look just right, and detecting problems when interfacing with another behavior. This dissertation's first proposal is a set of general-purpose visualization module designs for helping expose architecture state to authors during their performance evaluation phase. For those thinking of building or upgrading their architecture for more author-friendly debugging support, we highly recommend adding these prescriptive designs to your agent authoring architecture.

Reliable Triggering

My brother doubted that computer science was a proper "science" until he saw me debugging: applying the scientific method of having a hypothesis for the problem or solution; rigorously testing the code by changing variables or by holding variables constant to see if my hypothesis was valid; and then either applying the correct hypothesis or changing the hypothesis based on new information. An absolute necessity of proving a sound hypothesis based on experimentation is *repeatability*. The core of this module is providing a consistently repeatable path through the decision tree, whether it is to repeat the appearance of the bug or to repeat the setup for the bug in order to test if a change has fixed the bug. In order to execute this design, the architecture will need the following elements:

RT1: A meta representation of state and decisions
RT2: A means of automatically triggering decisions
RT3: Controlled randomness, if any randomness is used

In informal terms, reliable triggering (RT) can be seen as saving and loading a particular path through the architecture's behavior tree (or through each agent's personal behavior trees). RT1 represents the means by which this saving and loading can occur: the meta representation must contain relevant data that is not inherent to the behavior tree itself, and it may contain additional information. For some architectures, all they will need for RT1 is log of commands entered by the user that will be executed by RT2. If timing is important to the architecture, timestamps and encoded delays in executing the behaviors may be required. And if there is randomness, saving and using a random seed for RT3 would remove any remaining possible variation in reliably creating the same scenario. Any additional information on the state of the agents, for example, can be useful to the author if presented intelligently.

Implicit in all of these stored commands is a means by which the architecture can execute these commands. Some architectures may not be capable of this on their own, and may need some scripting wrapper or other interface to enable this meta functionality. However, the payoff for this functionality is enormous. Not only does it remove error and doubt in the scientific process of debugging agent behavior (allowing the author to remain focused on the task at hand), but it enables many potential instances of the agent to be executed simultaneously side-by-side. The author can then see the results of different hypothesis at the same time, rather than testing them sequentially. Additional information processing of the state and decision representation can highlight what aspects of the agents are inconsistent with one another in this side-by-side view. The following ABL-focused section will cash out what these advances would look like and mean for an ABL author.

Tweaking Performance Metrics

Each of these sections are meant to be isolated instances of improvement. However, this section on tweaking performance metrics (TPM) works in tandem with many elements of the previous section. In isolation, live TPM for the architecture offers increased author agency. It requires:

TPM1: Access to where (in the architecture) the metric is defined
TPM2: A means to change the metric, preferably in real time, with immediate results

The metrics that we are envisioning here are generally some numerical value, such as a weight from 0 to 1, or an enum. Given that the metrics are trying to be externally accessed, they should be stored as some global variable or in some kind of globally accessible (such as an indexed) data structure. The TPM2 element, such as a drop-down menu or a slider, would use its TPM1 access to immediately override the value. Instead of requiring the author to change code and recompile, the TPM would change the metric with a quicker and more immediate interface. If this is something the architecture needs to do often, some object-oriented design can make a reusable pattern for generating and hooking up the GUI element to the metric for the author with a single line of code.

Depending on the architecture, TPM can be more or less useful. For example, in our previous work section, TPM would be a much more important feature to add to the FAtiMA architecture than ABL or BOD/POSH. When interfacing with the previous RT section, each simultaneous agent instance could have a different interface for tweaking the metric. Thus, the author could simultaneously see the result of multiple instances of the metric altering the performance of the same set of behaviors, for example. This would rapidly decrease the time it would take for an author to narrow in on a value for the metric that they find pleasing.

Behavior Interfacing

Understanding how behaviors interact within an architecture is a more advanced requirement, usually found when many behaviors have been made and have to interlock in a pleasing manner. After the freshly authored behavior code is checked to compile, run properly, and functions well on its own, the last stage of the author's debugging task is often to verify that the behavior works well when interfacing with other code. What possible combinations of behaviors are possible is completely reliant on the architecture and the scenario. However, authors have requested a means to detect when behaviors interfere with one another, as well as methods of consistently and explicitly blending behaviors wherever possible. To satisfy these requests, we propose the following requirements:

BI1: Meta tracking of ongoing behaviors
BI2: High-level managers to mediate resource conflicts
BI3: Alerts for "hanging" or "stalling" behaviors

When debugging ABL code, the behavior tree can be visualized as a series of directory folders in a computer file management, where each `subgoal` or `spawngoal` corresponds to delving down another level in the hierarchy. Authors have expressed that seeing this whole hierarchy would be overwhelming. However, flagging certain specific behaviors using some meta language has been useful in the past (see related work on the ABL debugger). An alternative approach could be to select behaviors to "watch" where the author is alerted if the behaviors are interrupted. An interruption could be interpreted as any shift of focus from the "watched" behaviors (that is, execution of behavior steps not in the those behaviors) by the behavior tree execution algorithm via BI1.

Not only does behavior code execution need to be interfaced, but conflicting resource use is another common author challenge. Mateas' quintessential example for Façade is the blending the behaviors of drink-mixing and conversation-having with Trip's hands. Both behaviors involved using the hands, but they could not perform both tasks at once. In this case, ABL had both high-level managers to detect when parts of the body were occupied, as well as extra authored behaviors for Trip's body language as alternatives to using his hands when they were occupied. These behavior interfacing (BI) tactics were author-designed, but could be better supported by the architecture itself. Note that "resources" here does not necessarily mean bodily resources only: they could be any mutually exclusive architectural element.

Another example of BI difficulties can be shown via a lack of expected behavior interaction. In the previous work section, we showed an example behavior of an NPC offering an object to the player with no recourse if the behavior was left unanswered: the NPC would stand with his hand out for all eternity. A quick resolution to this problem may be to automatically abort the behavior after a timeout period, but the social repair of that broken interaction should be addressed for a believable character. Authors may not even think to address the broken behavior until much later, or not even know why an agent is stalling, as it may have been an untested edge case in a behavior authored days or weeks prior. A simple warning mechanism via BI3 can help find these edge cases much sooner.

Behavior Reuse

The previous section on visualization modules introduces and describes a number of general, somewhat high-level authoring challenges faced by previously interviewed authors, and a mixture of requests and proposals to lessen those challenges. However, many of these modules require a non-trivial amount of effort to support. Why bother? The most common problem we found in our previous work, at least mentioned by every team and institution, was the need for reusable behaviors. By some combination of modular design patterns, robustly authored and believable behaviors, and an interface to understand existing behaviors (likely authored by someone else): previous authors wanted to leverage previous work and have others make use of their behaviors. Since the previous section focused on robust behaviors, the following proposal outlines modular design patterns. The following section on the ABL authoring tool gives an example proposed interface.

Modular Design Pattern

In general, an author can try to apply principles of modular programming by minimizing mutual interaction with other modules and making self-contained behaviors. However, this is nearly impossible for behaviors in practice. Just previously there was an example of a "hanging" behavior to pass along an object to another agent: a behavior that required the presence of not only another behavior, but another agent, in order to finish. Interactivity in behaviors is a common occurrence because we *want* our agents to richly interact. The next best solution is to acknowledge, accept, and accommodate common forms of interaction between behaviors so that the forms of interaction are explicit for authors to create, evaluate, and reuse.

Let's work with an example behavior to examine these patterns: eye gaze. This behavior can be seen as simple as setting a location point of where an agent's eyes are focusing, or as complex as a one expression of agent attention (of which body orientation or head facing are other expressions). Some architectures may couple head facing and eye gaze together, as it is common for humans to orient both body parts together instead of having eyes rolling freely in their sockets. This behavior appears simple, but as soon as multiple points of interest are introduced, we have a conflict of what to look at: does the agent slowly alternate (and how so?), or try and find a medium between them?

Immediately we see that the idea of a modular behavior is a little misleading, in that we more accurately desire a reusable *set* of hierarchical behaviors, or a behavior module. In the next Figure 17, there is an example hierarchical eye gaze module, with each bubble representing a behavior. There are authored interfaces with the module for authors to make use of: getters and setters for other behaviors to use as explicit access points that the internal higher-level manager decides what to do with, as well as access points for the behaviors to use the animation engine (or whatever other behaviors it acts upon). Within the cloud, the author of the module is free to exercise whatever logic they desire to handle the inputs/outputs in an expected manner. However, incorporating a module manager for handling incoming access points and variations in execution is expected when following this pattern.

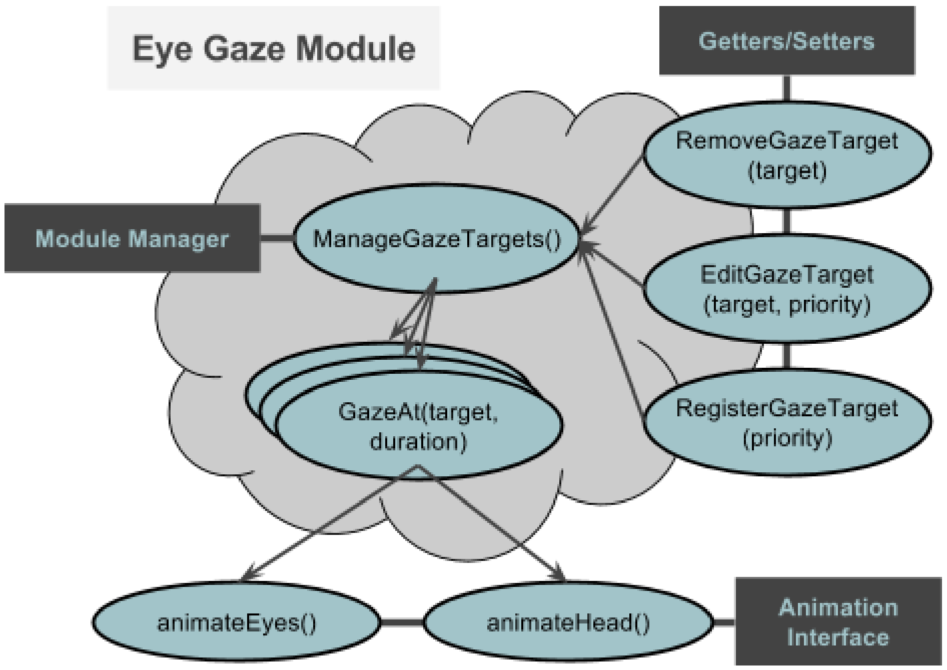


Figure 17: An example low-level behavior module. Includes interface points as getter/setter behaviors, a module manager within the heart of the module's logic, and a low-level connection to the animation engine.

Now imagine you were handed a completed eye gaze module: how would an author plan to use it? We envision the module like a code library for a regular programming language: a set of functionality that can be copy-pasted and hooked into existing behaviors with minimal effort. Authors could accept the module as-is, alter how the management code works, or flavor the GazeAt(target, duration) behavior with different moods or attitudes, for example. But not every module can be so cut-and-dry. Let's examine that item-giving example from above in the following Figure 18.

Complex modules begin to tangle very quickly, especially if there are other modules vying for the same resources. Higher-level modules (in this case, a step removed from nearly one-to-one mapping of the agent's body) make use of other modules, which in library terms is called a dependency. If an author were to use this item exchange module, they would require the dependency modules as well, and each would require the appropriate hooks into the author's animation and input systems. However, there are still clear connecting points between modules and sectioned-off pieces of code for reuse. Here is another example manager, this time orchestrating a sequence of behaviors and handling social repair if the sequence fails. Appropriate documentation, whether within the code or in a library repository, on the module's dependencies, expected use cases, and features would make these modules much more useable. Debugging and visualization information could also be filtered via these modules (rather than individual behaviors) and help isolate problems more quickly.

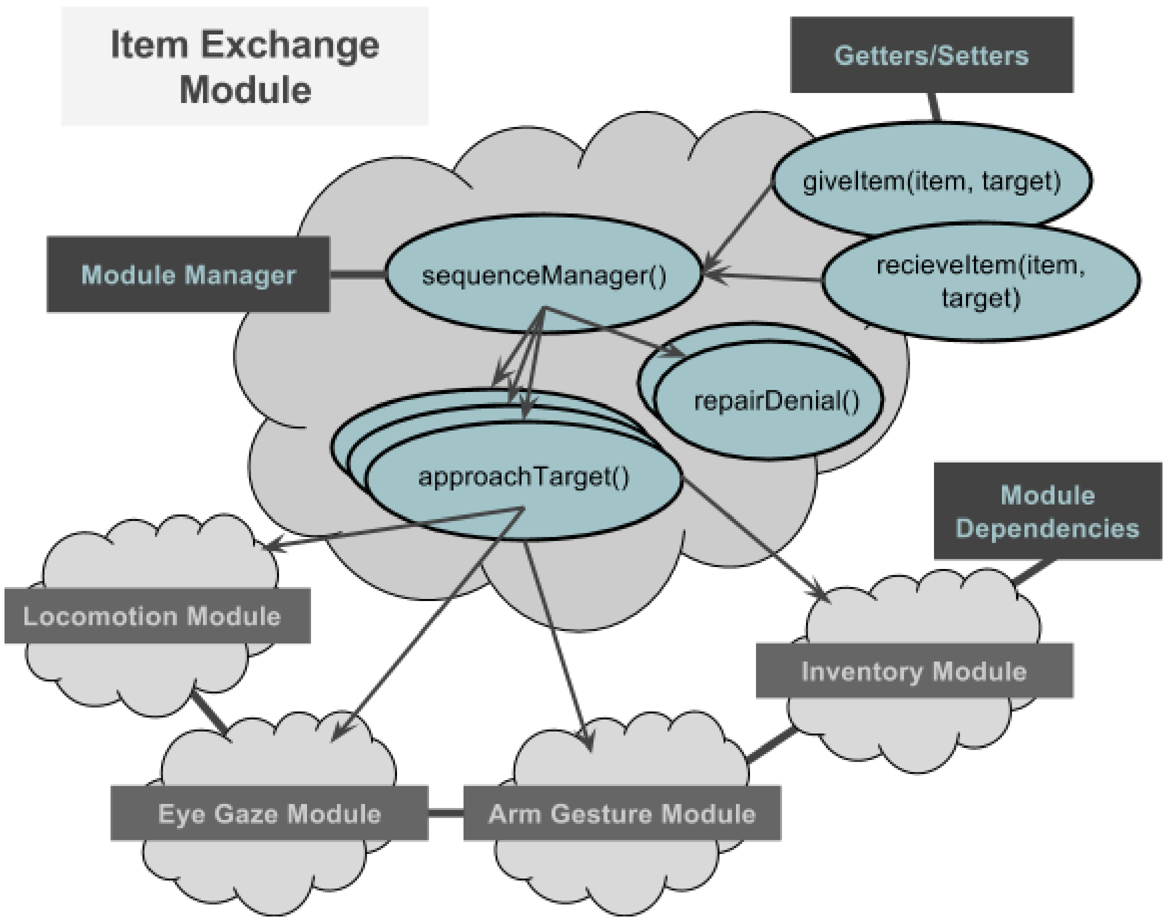


Figure 18: An example higher-level behavior module. It has the same getter/setter behaviors, as well as a manager. However, its functions call upon the getter/setter behaviors of other modules to actually pull off its sequence of requirements to pass the item.

ABL Authoring Interface, ENABL

The second main thrust of this proposal is applying all the theoretical proposals just discussed to the ABL agent architecture and its constructs. These theoretical proposals attempt to satisfy two primary purposes: easing the authorial complexity for ABL and reifying ABL idioms identified by previous ABL authors in the related work.

Definitions

First, a brief review of the definitions in the context of ABL. Mateas has stated that ABL has a high authorial burden (Mateas 2002), as it encompases both intent formation logic, behavioral selection planning, and behavioral execution with the *behavior* and *working memory element* (WME) constructs. ABL has no built-in higher-level concepts, and as such the hierarchical organization of behaviors is entirely up to the author's discretion to use or create. Combine this freedom with the multitude of features ABL offers the author in terms of specificity, context conditions, joint behaviors, and so on; and it is clear that ABL authors have extremely high leverage. However, ABL behaviors are also extremely complex to author.

Visualization Modules

Lessons learned from the Grail Framework SMDT taught us that authors desire sections of a technical design tool to be focused on accomplishing a specific task extremely well, rather than an open interface capable of solving every problem simultaneously. We broke up the Visualization Modules section with this design approach in mind, and we propose a separate section of the design tool to accommodate each visualization module.

Reliable Triggering

To reiterate the outlined requirements for this section:

RT1: A meta representation of state and decisions
RT2: A means of automatically triggering decisions
RT3: Controlled randomness, if any randomness is used

The ABL language would have to be extended to be cognisant of meta state and decisions (which are, in ABL's case, chosen behaviors), or some wrapper added around ABL's execution could detect these changes. The state and chosen behaviors can be stored in a simple text file and parsed and visualized for many creative purposes (see below). Whatever I/O ABL is situated within (whether command line or GUI interface) can be scripted to execute commands automatically. If ABL does not already use a random seed to pick between equally viable behaviors, it can be extended to do this as well.

The more interesting aspect of applying this design is envisioning an example use case. There are two approaches to generating the state/decisions text file: automatically recording playthroughs of the system in a robust fashion (ie saving the logs up to a crash) and scripting actions within the log (entering via a tool or by hand decisions the agent should take or changes to their state). The recording interface would be exceptionally simple. However, the state interface has the potential to be very complex (see Figure 19). If code libraries for ABL connections to I/O are made available, such as for Unity, modules for visualizing these agents can also be reused.

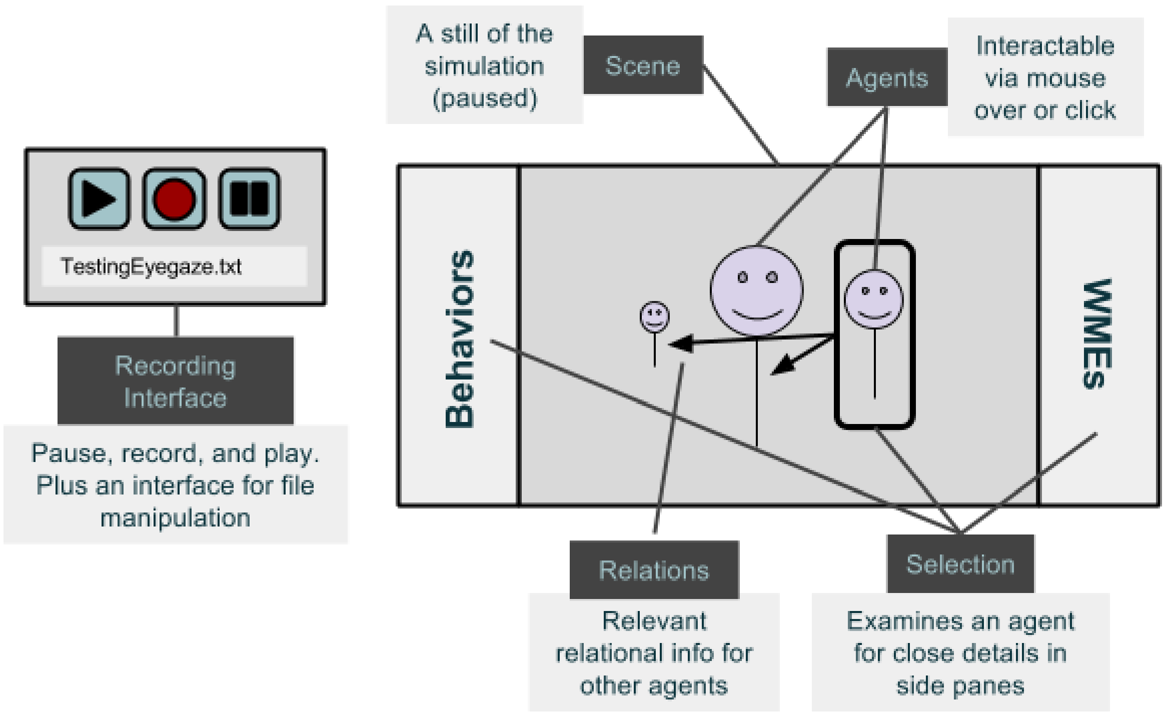

A still of the simulation (paused)
Scene
Agents
Interactable via mouse over or click
TestingEyegaze.txt
Recording Interface
Pause, record, and play. Plus an interface for file manipulation
Behaviors
WMEs
Relations
Relevant relational info for other agents
Selection
Examines an agent for close details in side panes

Figure 19: The reliable triggering graphical user interface (GUI). Includes an interactable interface over the scene that presents WMEs and behaviors filtered by agent and their current state.

One primary goal is to make the state-decision log completely independent so that many instances of the agent(s) can be run via the state-decision logs simultaneously. Each of these logs can have a slightly different script and be testing a series of common cases, similar to unit tests.

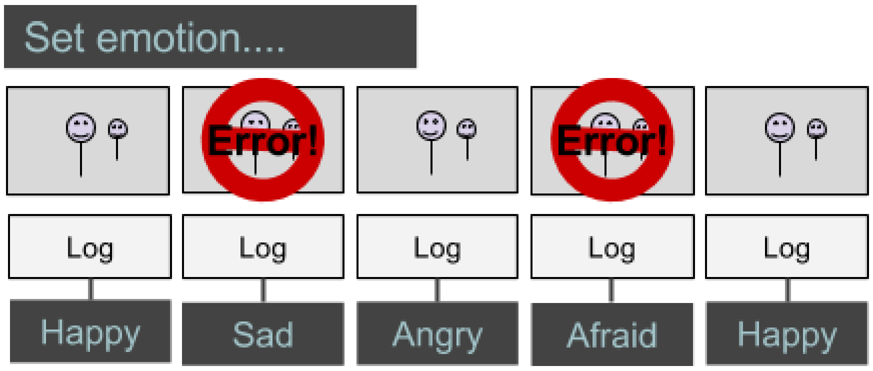


Figure 20: 5 simultaneous instances of the agents with different emotions. Here, two crash! Found in the time to run the agents once rather than 5 times sequentially.

Alternatively, a tree search algorithm could be applied to the ABL state space to procedurally generate test traces with the system and agents. These traces can then be automatically run and checked for crashes -- all independent of any authoring or testing by the author. While this cannot check for performance abnormalities, quality, or tweaking automatically, it helps with a chunk of the debugging process (see run-time errors of the authoring process Figure 12).

Tweaking Performance Metrics

To reiterate the outlined requirements for this section:

TPM1: Access to where (in the architecture) the metric is defined
TPM2: A means to change the metric, preferably in real time, with immediate results

In addition to the recording interface, a small slider interface can be added (TPM1, see FIGURE for example). ABL variables are already easily made either global or stored in a global-space WME (TPM2), and so long as they are accessed in real-time via behaviors, they can offer immediate results.

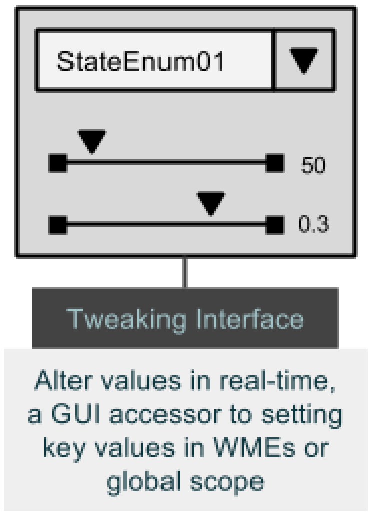

StateEnum01
50
0.3
Tweaking Interface
Alter values in real-time, a GUI accessor to setting key values in WMEs or global scope

Figure 21: Enums or variables that need to be tweaked or that express different behaviors should be easy to edit.

Behavior Interfacing

Once again, the outlined requirements:

BI1: Meta tracking of ongoing behaviors
BI2: High-level managers to mediate resource conflicts
BI3: Alerts for "hanging" or "stalling" behaviors

In the RT FIGURE above, we snuck in a visualization pane for the ongoing behaviors filtered for a specific agent when they were selected because of how common the request for that information was by ABL authors. The tracking of such information is fairly easy in ABL's architecture, and a small internal ABL tool was built by Larry LeBron to examine all ongoing behaviors in ABL's ABT. We plan to consult with LeBron to expand and improve this tool for general use for the ABL language.

Defining the modular design pattern with a manager meshes with the B12 requirement nicely, as a manager is built into every module. This manager is meant to handle resource conflicts, blending resources wherever possible and repairing failed attempts at behavior or sequence performance. Having an explicit area in the modular design pattern for B12 code helps promote its use by authors. As for BI3, ABL has the specific `wait` command that is often the culprit for "hanging" behaviors. Although the author clearly used the `wait` command for a purpose, the specific active `wait` commands that an actor is executing are easy to find. Once found, the author can re-evaluate whether the command is performing as intended and change their approach accordingly.

While not directly related to BI per say, the tracking of ongoing behaviors past and present allows the gathering of statistical data on the traversal of the ABT possibility space (also known as the dependency tree). Going back to those generated and automated runs from RT1 and RT2, we can see statistics on how often different behaviors are actually used: valuable data that current ABL authors can only get a "feel" for. For example, if a "sad" variation is never used, the author can examine if this is an issue with the specific behavior, or a problem with whether the agent is ever made sad. However, without seeing that statistic, the author may have forgotten to test the "sad" variation and never known it was broken.

Behavior Reuse

The previous sections have demonstrated how the visualization modules would be implemented in ABL, as well as example uses for their features. These modules have the primary purpose of easing the authorial complexity for the author. The modular design pattern focuses on reifying the most common ABL idioms. With a consistent structure and consistent idiom pattern, different authors can then share their authored behavior modules, and the long-awaited goal of reusable behaviors is within reach!

Modular Design Pattern

While defined as a single pattern idea for generality's sake above, ABL will see different flavors of the behavior module. Two of these flavors have already seen examples: the low-level module that is close to body resources, and the high-level module that orchestrates multiple low-level modules. Let us examine these examples more closely to see where ABL idioms manifest.

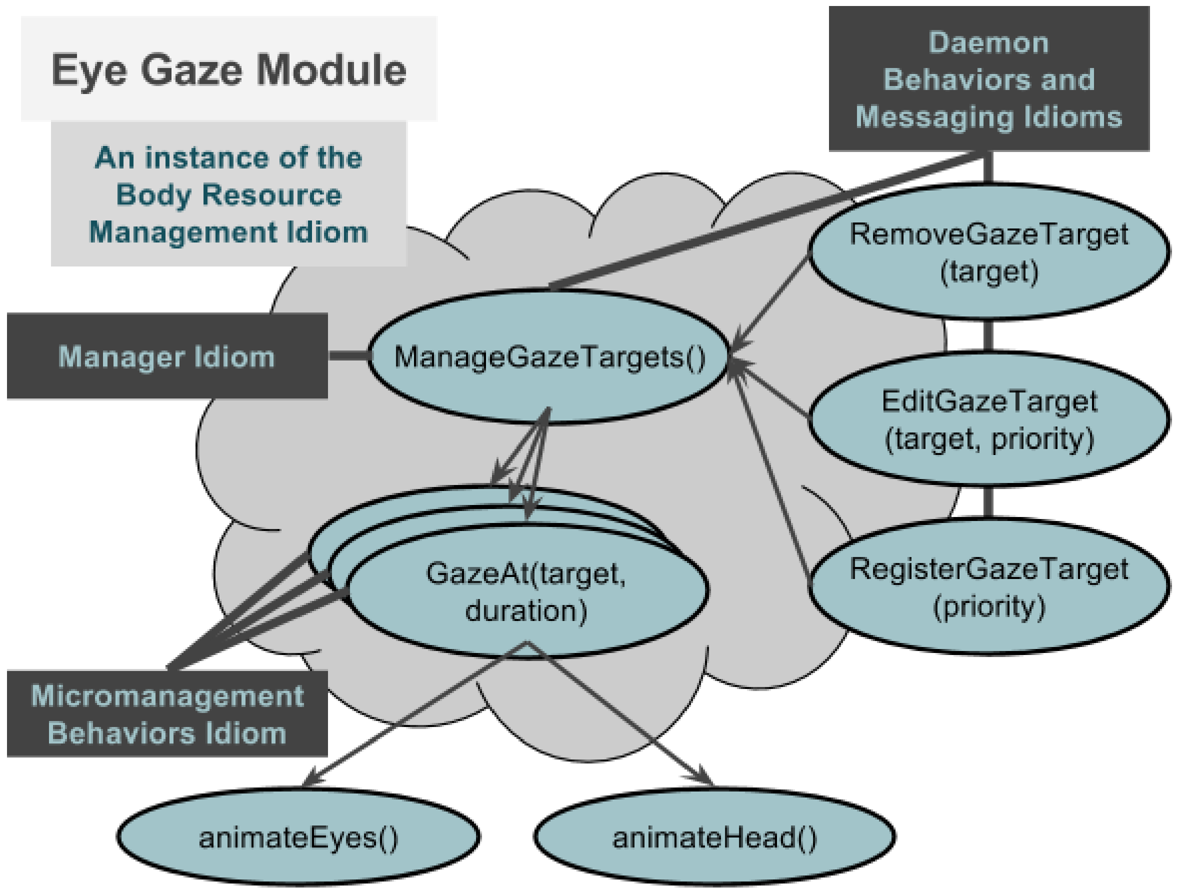

Eye Gaze Module
An instance of the Body Resource Management Idiom
Daemon Behaviors and Messaging Idioms
Manager Idiom
ManageGazeTargets()
RemoveGazeTarget (target)
EditGazeTarget (target, priority)
RegisterGazeTarget (priority)
GazeAt(target, duration)
Micromanagement Behaviors Idiom
animateEyes()
animateHead()

Figure 22: Figure 17 rephrased using more ABL-specific idioms.

The general structure of the modular design pattern (the input/output specification, the inclusion of a manager, and the helper behaviors) represent all the idioms that Weber et. al. outlined in their paper (Weber et. al. 2010). The manager behavior's existence exemplifies the manager idiom. The manager should, if it is accepting command messages like those in the eye gaze module, be a daemon behavior, as those two idioms come in a pair. The behaviors that the manager calls once it has decided how to act on its given messages are a direct example of the micromanagement behaviors idiom. These idioms, which belong to both the eye gaze module and the item exchange module (see Figure 22), are reified via adherence to the modular design pattern by necessity.

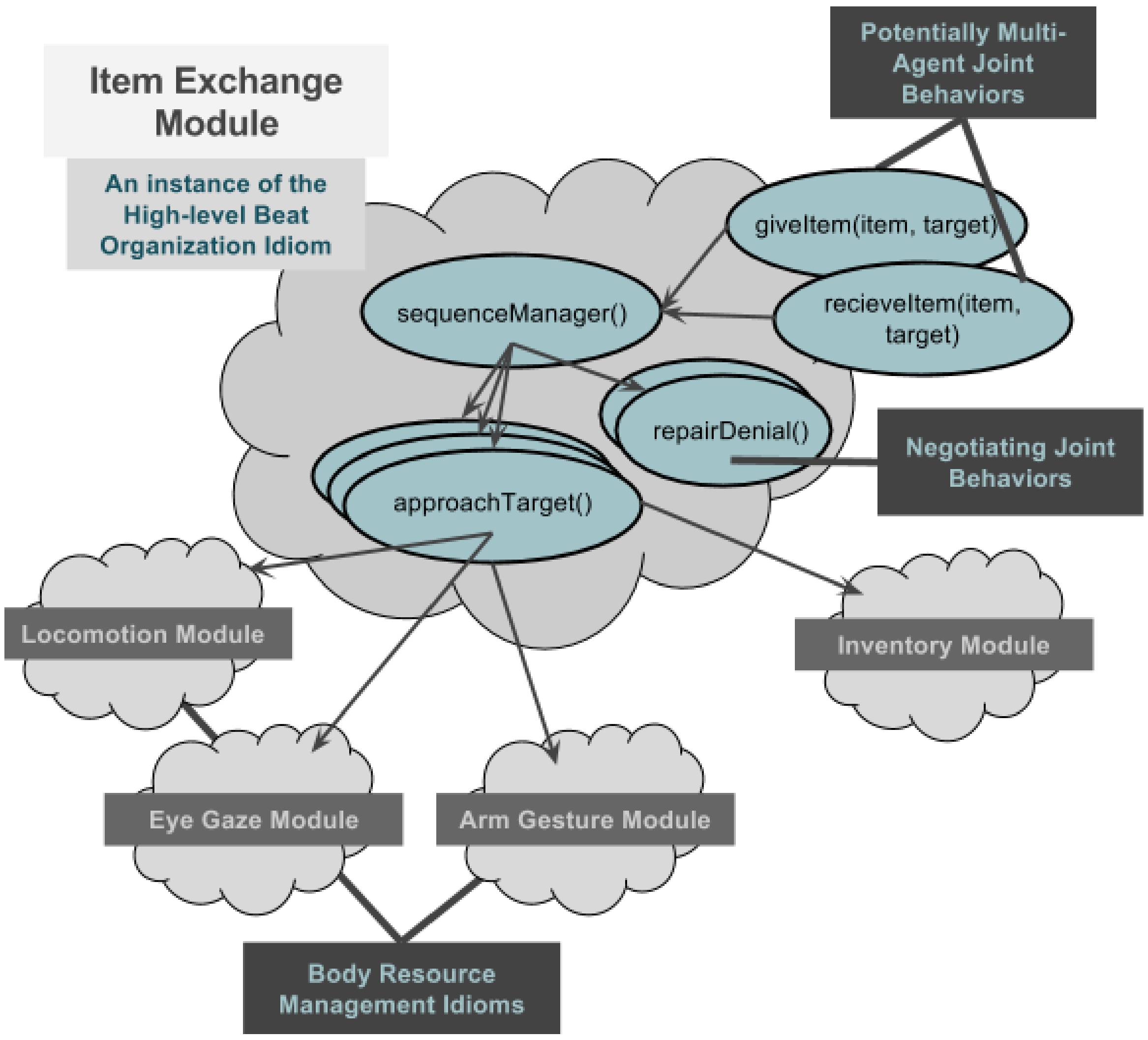

Item Exchange Module
An instance of the High-level Beat Organization Idiom
Potentially Multi-Agent Joint Behaviors
giveItem(item, target)
sequenceManager()
recieveItem(item, target)
repairDenial()
Negotiating Joint Behaviors
approachTarget()
Locomotion Module
Inventory Module
Eye Gaze Module
Arm Gesture Module
Body Resource Management Idioms

Figure 23: Figure 18 rephrased using more ABL-specific idioms.

Mateas & Stern's example idioms are reified via different executions of this design pattern for different purposes. For example, this low-level eye gaze module is what they would call a body resource management idiom because its manager's primary function is to mediate body resource conflicts about what the agent should gaze at. The hypothetical arm gesture module and locomotion module from the item exchange module (see Figure 23) are also examples of this body resource management idiom. The item exchange module itself, however, would be an instance of the higher-level beat organization idiom, as it is orchestrating a series of specific interactions.

Depending on the approach of the ABL author, the item exchange module is also a potential example of joint behaviors. Two agents may enter the item exchange module, one with the command to give an item, and another with the command to receive it. The manager attempts to orchestrate the two characters together for the exchange of the item, accessing the agent's body resource idioms to accomplish this. However, if one or both agents are unable to participate (whether by physical or social constraints), a series of behaviors is waiting to be executed by the manager to handle a variety of negotiation failures. These behaviors may in turn call other modules, such as a Social Faux Pas module that determines how the agents handle the failed negotiation and if they should try again.

The general modular design pattern also aligns with the Social Interaction Unit idiom that IMMERSE's ABL authors employ (Shapiro et. al. 2013). Because their ABT holds all agents, not just a single agent, their manager involves much more role specification between initiators and responders of the behavior. The performance manager, which aligns and orchestrates all the other behaviors, acts as a monolithic manager at the highest level of the ABT hierarchy: in other words, it acts as an all-encompassing module that contains many modules within it. This is why we gave the caveat of "depending on the approach of the ABL author" above -- IMMERSE's approach of holding all agents within one ABT means that no joint behavior negotiation needs to happen *between trees*.

Reifying the Idioms

The modular design pattern holds up to various example ABL behaviors that an author may want to create, including at different levels of hierarchical detail and for varying levels of complexity. However, we propose enumerating these patterns for an author and giving them a starting point *with code*. The author doesn't alter the structure of the given code too much, the ABL behaviors they write will more likely be a great candidate for use as an ABL library for code reuse! The following Table 4 shows example idioms we have discovered so far, a brief description of their purpose or functionality, and example instances of the idiom an author may create (which we will build templates for).

Name	Planned Support	Examples
Daemon Behaviors (Weber et. al. 2010)	An on-going listener for messages. Every Body Management module will likely be a Daemon.	manageGazeTargets, manageHeldItem, manageLocomotion
Messaging (Weber et. al. 2010)	A behavior that subgoals or spawns a goal for Daemon Behaviors. Getters/Setters for Daemons.	setGazeTarget, moveToLocation, pickUpItem
Managers (Weber et. al. 2010)	The standard template for the management behavior within each module. Likely Daemon.	manageGazeTargets, manageLocomotion, itemExchangeSequencer
Micromanagement Behaviors (Weber et. al. 2010)	Lowest level behaviors that call direct agent action.	gazeAt, move, setArmPosition, playAnimation
Higher-level Beat Behavior Organization (Mateas & Stern 2004)	The term for managers when used for higher level behaviors (ie not body resource management).	itemExchangeSequencer, attentionManager, socialEtiquetteManager
Body Resource Management (Mateas 2002)	The term for managers when used for lower-level behaviors (ie not beat behavior organization). Likely Daemons.	manageGazeTargets, manageLocomotion, manageArms

Input-Handling Behaviors (Mateas & Stern 2004)	Behaviors that interpret sensor information from the player/interface. Can be simple or very complex, and often trigger agent responses.	findTarget, estimateEmotion, parseNaturalLanguage
Joint Behavior Negotiation (Mateas 2002)	Behaviors that orchestrate actions from multiple agents and signal managers for failure recovery if necessary. Likely Daemons.	giveItem/recieveItem, offerToDance/ acceptOfferToDance
Multi-Agent ABT Negotiation (Shapiro et. al. 2013)	The Joint behavior negotiation for ABT's that hold all agents. Uses different plumbing to work within the tree.	giveItem/recieveItem, offerToDance/ acceptOfferToDance
Mood/Emotion Wrap-ons (Shapiro et. al. 2013)	Additions to Body Resource Management that flavor Micromanagement Behaviors.	gazeAngrilyAtTarget, gazeCuriouslyAtTarget
Performance Manager (Shapiro et. al. 2013)	The Highest level Beat Behavior Organization. Directs scenario-level instructions for a specific sequence or as a drama manager. A Daemon.	act1, act2, act3, generateConflict, denouement, interruptPerformance

Table 4: A summary of ABL idioms defined to-date, as well as their general purpose and examples for how authors may use them. Some of them overlap or make use of each other, as they have been defined by different authors working toward different bodies of work.

Evaluation

Applying Authoring Patterns

The first section of our proposed work was a general set of authoring support strategies and patterns to be used by any agent architecture. The second section is our evaluation for the first: a proof-of-concept application of the strategies and patterns. We plan to publish these strategies and patterns, as well as their example implementation, as a guidebook for other agent architectures to follow. The proposed work outlined how we would apply each proposed idea to the ABL architecture. The next sections in this evaluation will show how we plan to evaluate the architecture that implements these patterns.

Authorial Leverage

In the related work, we saw how Chen et. al. evaluated authorial leverage for drama managers by examining different qualities of authored decision trees (Chen et. al. 2007). Some of these evaluations were possible because they built upon previous work that established, "the evaluation function can correspond with expert evaluations of experience quality" by Weyhrauch (Weyhrauch 1997). There is no such evaluation function for simulated users interacting with agents. In any case where *agent* quality must be measured, we must either build our own evaluation function or use quantitative results from a user study. Examples from the related work section involve methods of evaluating believability of agents that we can use for this. However, we are focused on the quality and ease of authoring experience, as well as the quantity of output in a given amount of time, in this proposal and not on guaranteeing quality agents. So long as two agents can be judged to be of equivalent quality by performing the same tasks in similar manners, their behavior trees can be suitably analyzed and compared.

Complexity: Examining the smallest tree that achieves reasonable performance and qualitatively examining whether it would be reasonable to hand-author. Or, approaching from the other direction, beginning with a small ABT and measuring how complexity grows when adding additional behaviors or behavior modules. (Quantity)

Ease of policy change: Assuming a fully realized tree, how difficult is it to tune and alter the agent's behaviors? Standard ABL tree edits require many phases of debugging, while the visualization modules shorten or eliminate many of these phases for the same policy change task. (Quality)

Variability of experiences: Examines the variety of agent behavior by measuring the frequency of variability. Attempts to capture a breadth of different high-quality interactions with the agent (Quantity & Quality)

These three metrics capture the essence of ABL authorial leverage while taking into account ABL's currently massive authorial complexity. We plan to examine the output of the following user studies using these metrics in order to quantitatively measure the changes to ABL's authoring experience before and after applying the proposed authoring strategies and patterns.

Usability & User Studies

The previous incarnations of ABL's authoring tool have taught us that if the tool is not usable, authors will not use it, regardless of its perceived benefits. If the proposed patterns are too difficult to make use of, or the interfaces too fragile and buggy, they will not be used. All of the promises in this proposal would be for naught. We propose a series of iterative and rigorous user studies in order to prevent this scenario from occurring. We will present expert, intermediate, and novice ABL authors with progressive instances of the proposed patterns and tools and judge their effectiveness for different audiences. As expert ABL authors are housed at UCSC and in close proximity to this proposal's author, we plan to deploy advances of our tool to them for long-term case studies on their usability and effectiveness in large-scale projects.

We also understand that expert ABL authors may not wish to use many of the tools, as they have their own shortcuts or patterns too specific for ABL to be made general in a proposal such as this one, or that novices may need a much richer starting ABT in order to begin authoring their own agents by example. We will do our best to prepare each user study accordingly to maximize the author's leverage and the successful creation of their agents, as well as maintain consistency within author experience groups.

Usability

The field of Usability Testing has standards for user interface evaluation that we can take advantage of. In general, the user studies involve the interviewer examining a user attempting to use the interface for its intended purpose, often gathering UI operations and questionnaires or surveys before and after the study. The study itself involves completing a number of tasks using the given interface. The exact list of tasks will be dependent on what aspects of the proposal design have been completed, but will ultimately involve editing or creating behaviors or ABL agents. The testing protocols will involve a mixture of the following (whatever is possible depending on location and subjects):

Thinking-aloud Protocol: user talks during the test
Question-asking Protocol: tester asks user questions during the test
Performance Measurement: tester or software records usage data during test
Log File Analysis: test analyzes usage data
Co-discovery Learning: two users collaborate
Interviews: one user participates in a discussion session
Surveys: interviewer asks user specific questions
Questionnaires: user provides answers to specific questions

(Ivory 2001)

Authored Agents

Using the data gathered in the usability studies, we can apply the authorial leverage metrics, both of which were described above. We will be careful to classify gathered data by the version of the tools presented to the users, their length of time interacting with the tool, their programming experience, and their experience with ABL and other agent architectures.

Target User Groups

As previously mentioned, we have ABL experts (as well as intermediate and novices) housed at UCSC for short- and long-term user studies. However, we also have plans to export these ABL tools to other authors of ABL that have moved on from UCSC, such as Gillian Smith and her students, who are interested in a more user-friendly ABL experience. To further push the concept of iterative design and introducing author feedback as early as possible, we plan to host workshops at a variety of conferences with a focus on agent authoring interfaces (see the following schedule for a detailed list). At these workshops, we will present attendees with the most recent version of our work and conduct group user studies with them; this is where we imagine the Co-discovery Learning protocol from above to come into play. Finally, as the authoring experience becomes more polished, we plan to push the user studies to even more novice users unfamiliar with agent authoring at all by hosting a seminar or class at UCSC where undergraduate students will make ABL agents.

Schedule

2014	Fall	ABL extension design & execution ABL dependency tree analysis (with Swen Gaudl)
2015	Winter	Prototype scripting commands for ABL's extensions Prototype recording interface IUI Workshop* (possibly with Swen Gaudl) AAAI Workshop* (possibly with Swen Gaudl)
	Spring	Prototype scene display with Unity FDG Workshop*
	Summer	Automatic ABT exploration algorithm ABT exploration statistics

		ICIDS Workshop*
		Creativity & Cognition Workshop*
		ICCS Workshop*
	Fall	Build first draft of authoring library via authoring patterns
		IVA Workshop*
		DiGRA Workshop*
		AIIDE Workshop*
2016	Winter	Revise tools, displays, and libraries for final round of studies
		Begin Dissertation writing
		IUI Workshop II*
		AAAI Workshop II*
	Spring	Present full suite of tools to authors for final rounds of studies
		Propose/teach class on ABL authoring
		FDG Workshop II*
		Continue Dissertation writing
		Begin Job Search
	Summer	Propose/teach class on ABL authoring (if Spring is not possible)
		Continue Dissertation writing
		Job Search
	Fall	Defend Dissertation

* Note: These are assuming these conferences are occurring at all, are occurring at roughly at times of the year where they have previously, and that the workshop proposal is accepted.

Conclusion

This document has presented a detailed plan for enabling the authoring of dramatic, embodied, and interactive virtual agents, as well as a schedule for executing that plan using the ABL architecture as a baseline. First, a history of the type of agents targeted by this plan, as well as their behavior trees and previous incarnations are explored via related work and previous work. Next, the plan for supporting agent authoring is explained: it includes general specifications for common authoring patterns and visualization modules designed to target specific problems authors face while following those patterns. Application of this plan is then outlined for the ABL architecture, and benefits to authoring are hypothesized for future testing. Finally, analyses of authoring and tool usability testing are described as future plans for evaluating the resulting authoring interface.

The goal of this body is work is to show that an explicitly modular authoring approach can reduce authorial complexity in creating dramatic agents, which are extremely complex by necessity. Thus, the authorial leverage of agent authors is increased, and more users can easily create agents for whatever purpose they desire: be it games, education, or any of the many other fields creating virtual agents. While the work in this proposal focuses on ABL agents, the proposed designs can be extrapolated to any behavior-tree-based agent architecture (and possibly even further than that). Whether the world sees more ABL agents, or more agents of some other architecture, this author would be greatly satisfied so long as the authorial burden is no longer so frightening to new agent authors.

References

Agre, Philip. 1997. *Computation and human experience*. Cambridge University Press.

Aristotle. 330 BC. *The Poetics*. Mineola, New York: Dover 1997.

Becroft, David, et al. 2011. "AIPaint: A Sketch-Based Behavior Tree Authoring Tool." *AIIDE*.

Bates, Joseph. 1992. *The nature of characters in interactive worlds and the Oz project*. Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Bates, Joseph. 1994. "The role of emotion in believable agents." *Communications of the ACM* 37.7. 122-125.

Bates, Joseph, A. 1994. Bryan Loyall, and W. Scott Reilly. *An architecture for action, emotion, and social behavior*. Springer Berlin Heidelberg.

Bernardini, Sara, and Kaska Porayska-Pomsta. 2013. "Planning-Based Social Partners for Children with Autism." *ICAPS*.

Blizzard Entertainment. 1998. *StarCraft*. Blizzard Entertainment.

Blumberg, Bruce Mitchell. 1996. *Old tricks, new dogs: ethology and interactive creatures*. Diss. Massachusetts Institute of Technology.

Bone, Jeff. 2002. “Does XML Suck? Revisited.” *O’Reilly*. http://archive.oreilly.com/pub/post/does_xml_suck_revisited.html [Accessed 17 September 2014]

Brom, Cyril, et al. 2006. "Posh tools for game agent development by students and non-programmers." *The Nineth International Computer Games Conference: AI, Mobile, Educational and Serious Games*. University of Bath.

Bryson, Joanna J. 2003. "The behavior-oriented design of modular agent intelligence." *Agent technologies, infrastructures, tools, and applications for e-services*. Springer Berlin Heidelberg. 61-76.

Bryson, Joanna J., and Lynn Andrea Stein. 2001. "Modularity and design in reactive intelligence." *International Joint Conference on Artificial Intelligence*. Vol. 17. No. 1. Lawrence Erlbaum Associates LTD.

Cassell, Justine, et al. 1999. "Embodiment in conversational interfaces: Rea." *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM.

Champandard, Alex. 2007. “Understanding Behavior Trees.” *AIGameDev*. http://aigamedev.com/open/article/bt-overview/ [Accessed 17 September 2014]

Chen, Sherol, et al. 2009. "Evaluating the Authorial Leverage of Drama Management." *AAAI Spring Symposium: Intelligent Narrative Technologies II*.

Dias, Joao, Samuel Mascarenhas, and Ana Paiva. 2011. "Fatima modular: Towards an agent architecture with a generic appraisal framework." *Proceedings of the International Workshop on Standards for Emotion Modeling*.

Digman, John M. 1990. "Personality structure: Emergence of the five-factor model." *Annual review of psychology* 41.1. 417-440.

Dromey, R. Geoff. 2001. “Genetic Software Engineering - simplifying design using requirements integration.” In *IEEE Working Conf. on Complex Dynamic Systems Architecture*, Brisbane, Australia.

Dromey, R. Geoff. 2003. "From requirements to design: Formalizing the key steps." *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on*. IEEE.

Egri, Lajos. 1960. *The Art of Dramatic Writing: Its Basis in the Creative Interpretation of Human Motives. With an Introd. by Gilbert Miller*. Simon and Schuster.

Ekman, Paul, and Wallace V. Friesen. 2003. *Unmasking the face: A guide to recognizing emotions from facial clues*. Ishk.

Evans, Richard. “Computer models of social practices.” In *Synthese Library: Philosophy and Theory of Artificial Intelligence*. Synthesis, forthcoming.

Firby, R. James. 1987. "An investigation into reactive planning in complex domains." *AAAI*. Vol. 87.

Gebhard, Patrick et. al. 2003. “Authoring Scenes for Adaptive, Interactive Performances” In: *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems* (*AAMAS*'03), Melbourne.

Georgeff, Michael P., and Amy L. Lansky. 1987. "Reactive reasoning and planning." *AAAI*. Vol. 87.

Georgeff, Michael P., Amy L. Lansky, and Marcel J. Schoppers. 1987. *Reasoning and planning in dynamic domains: An experiment with a mobile robot*. SRI International, Menlo Park, CA.

Grow, April M., et al. 2014. “A Methodology for Requirements Analysis of AI Architecture Authoring Tools.” *Foundations of Digital Games ‘14*.

Gomes, Paulo, et al. 2013. "Metrics for Character Believability in Interactive Narrative." *Interactive Storytelling*. Springer International Publishing. 223-228.

Hayes-Roth, Barbara, and Patrick Doyle. 1998. "Animate characters." *Autonomous agents and multi-agent systems* 1.2. 195-230.

Heckel, Frederick William Poe, G. Michael Youngblood, and Nikhil S. Ketkar. 2010. "Representational complexity of reactive agents." *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE.

Hecker, Chris. 2009. "My liner notes for spore/Spore Behavior Tree Docs." http://chrishecker.com/My_liner_notes_for_spore/Spore_Behavior_Tree_Docs [Accessed 17 September 2014]

Huang, Zhisheng, Anton Eliëns, and Cees Visser. 2003. "XSTEP: An XML-based Markup Language for Embodied Agents." *Proc. 16th International Conf. on Computer Animation and Social Agents,(CASA 2003)*.

Ihnatowicz, Edward. 1986. *Cybernetic art: a personal statement*. Self-published. Available from: http://monoskop.org/Edward_Ihnatowicz [Accessed 17 September 2014]

Isbister, Katherine, and Patrick Doyle. 2002. "Design and evaluation of embodied conversational agents: A proposed taxonomy." *The First International Joint Conference on Autonomous Agents & Multi-Agent Systems*.

Isla, Damian. 2005. "GDC 2005 Proceeding: Handling Complexity in the *Halo 2* AI." *Gamasutra*. http://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php [Accessed 17 September 2014]

Ivory, Melody Yvette. 2001. *An empirical foundation for automated web interface evaluation*. Diss. University of California, Berkeley.

Johansen, Emil. 2014. "The Behave Project." *AngryAnt*. http://angryant.com/behave/ [Accessed 17 September 2014]

Kelso, Margaret Thomas, Peter Weyhrauch, and Joseph Bates. 1993. "Dramatic presence." *Presence: The Journal of Teleoperators and Virtual Environments* 2.1. 1-15.

Kopp, Stefan, et al. 2006. "Towards a common framework for multimodal generation: The behavior markup language." *Intelligent virtual agents*. Springer Berlin Heidelberg.

Krahulik, Mike, and Jerry Holkins. 2013 "Audacity." *Penny Arcade*. Web. http://penny-arcade.com/comic/2013/06/19 [Accessed 17 September 2014]

Lester, James C., and Brian A. Stone. 1997. "Increasing believability in animated pedagogical agents." *Proceedings of the first international conference on Autonomous agents*. ACM.

Lindsay, Peter A. 2010. "Behavior trees: from systems engineering to software engineering." *Software Engineering and Formal Methods (SEFM), 2010 8th IEEE International Conference on*. IEEE.

Loyall, A. Bryan. 1997. *Believable agents: building interactive personalities*. Diss. Stanford University.

Loyall, A. Bryan and Joseph Bates. 1991. *Hap: A Reactive, Adaptive Architecture for Agents.* Technical Report CMU-CS-91-147, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Madison, Dan. 2005. *Process mapping, process improvement, and process management: a practical guide for enhancing work and information flow*. Paton Professional.

Maiden, Neil, Alexis Gizikis, and Suzanne Robertson. 2004. "Provoking creativity: Imagine what your requirements could be like." *Software, IEEE* 21.5. 68-75.

Mateas, Michael. 1999. *An Oz-centric review of interactive drama and believable agents*. Springer Berlin Heidelberg.

Mateas, Michael. 2002. "Interactive drama, art and artificial intelligence." Diss. Carnegie Mellon University.

Mateas, Michael, and Andrew Stern. 2003. "Façade: An experiment in building a fully-realized interactive drama." *Game Developers Conference*.

Mateas, Michael, and Andrew Stern. 2004. "A Behavior Language: Joint action and behavioral idioms." *Life-Like Characters*. Springer Berlin Heidelberg. 135-161.

Mateas, Michael, and Andrew Stern. Released 2005. *Façade*. Procedural Arts. http://www.interactivestory.net/ [Accessed 17 September 2014]

Mateas, Michael, and Andrew Stern. 2005b. "Structuring Content in the Façade Interactive Drama Architecture." *AIIDE*.

McCoy, Joshua, and Michael Mateas. 2008. "An Integrated Agent for Playing Real-Time Strategy Games." *AAAI*. Vol. 8.

McCoy, Joshua, et al. 2011. "Comme il Faut: A System for Authoring Playable Social Models." *AIIDE*.

McKee, Robert. 1997. *Substance, Structure, Style, and the Principles of Screenwriting*. New York: HarperCollins.

Myers, Isabel Briggs.1962. "The Myers-Briggs Type Indicator: Manual (1962)."

O'Keefe, Sarah. 2010. "XML: The death of creativity in technical writing?" *Scriptorium*. http://www.scriptorium.com/2010/02/xml-the-death-of-creativity-in-technical-writing/ [Accessed 17 September 2014]

Ortony, Andrew. 2002."On making believable emotional agents believable." *Trappl et al.(Eds.)(2002)*. 189-211.

Pedica, Claudio. 2009. "Spontaneous avatar behaviour for social territoriality." Master's Thesis. Reykjavik University.

Perlin, Ken, and Athomas Goldberg. 1996. "Improv: A system for scripting interactive actors in virtual worlds." *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM.

Powell, Daniel. 2010. "Behavior engineering-a scalable modeling and analysis method." *Software Engineering and Formal Methods (SEFM), 2010 8th IEEE International Conference on*. IEEE.

Rao, Anand S., and Michael P. Georgeff. 1995. "BDI Agents: From Theory to Practice." *ICMAS*. Vol. 95.

Reed, Aaron A., et al. 2011. "A Step Towards the Future of Role-Playing Games: The SpyFeet Mobile RPG Project." *AIIDE*.

Reilly, Neal, and W. Scott. 1997. "A methodology for building believable social agents." *Proceedings of the first international conference on Autonomous agents*. ACM.

Riedl, Mark O., and Andrew Stern. 2006. "Believable agents and intelligent scenario direction for social and cultural leadership training." *Proceedings of the 15th Conference on Behavior Representation in Modeling and Simulation*.

Shapiro, Daniel G., et al. 2013. "Creating Playable Social Experiences through Whole-Body Interaction with Virtual Characters." *AIIDE*.

Simpson, Chris. 2014. "Behavior trees for AI: How they work." *Gamasutra*. http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php [Accessed 17 September 2014].

Smith, Gillian, Jim Whitehead, and Michael Mateas. 2010. "Tanagra: A mixed-initiative level design tool." *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. ACM, 2010.

Smith, Gillian, et al. "Launchpad: A rhythm-based level generator for 2-d platformers." *Computational Intelligence and AI in Games, IEEE Transactions on* 3.1 (2011): 1-16.

Sosa, Ricardo, and John Gero. 2003. "Design and change: a model of situated creativity." *University of Sydney, Sydney*.

Stone, Brian A., and James C. Lester. 1996. "Dynamically sequencing an animated pedagogical agent." *AAAI/IAAI, Vol. 1*.

Sullivan, Anne. 2012. *The Grail Framework: Making Stories Playable on Three Levels in CRPGs*. Diss. University of California, Santa Cruz.

Swartout, William, et al. 2010. "Ada and Grace: Toward realistic and engaging virtual museum guides." *Intelligent Virtual Agents*. Springer Berlin Heidelberg.

Traum, David, et al. 2012."Ada and Grace: Direct interaction with museum visitors." *Intelligent Virtual Agents*. Springer Berlin Heidelberg.

TSR. 1977. *Character Record Sheets*. *Dungeons & Dragons.*

Turing, Alan M. 1950. "Computing machinery and intelligence." *Mind*: 433-460.

van der Hoek, Wiebe, and Michael Wooldridge. 2012. "Logics for Multiagent Systems." *AI Magazine* 33.3.92.

Vilhjálmsson, Hannes, et al. 2007. "The behavior markup language: Recent developments and challenges." *Intelligent virtual agents*. Springer Berlin Heidelberg.

Wardrip-Fruin, Noah. 2009. *Expressive Processing: Digital fictions, computer games, and software studies*. MIT press.

Weber, Ben George, Michael Mateas, and Arnav Jhala. 2010. "Applying Goal-Driven Autonomy to StarCraft." *AIIDE*.

Weber, Ben George, et al. 2010. "Reactive planning idioms for multi-scale game AI." *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE.

Weizenbaum, Joseph. 1979. *Computer power and human reason*. Freeman.

Wen, Larry, and Geoff Dromey. 2009. "A hierarchical architecture for modeling complex software intensive systems using behavior trees." *9th Asia-Pacific Complex Systems Conference*. Chuo University.

Weyhrauch, Peter. 1997. "Guiding Interactive Drama." Diss, Tech report CMU-CS-97- 109, Carnegie Mellon University.

Wooldridge, Michael, and Paul E. Dunne. 2005. "The complexity of agent design problems: Determinism and history dependence." *Annals of Mathematics and Artificial Intelligence* 45.3-4. 343-371.