Developing Object-Oriented Frameworks For

Computer Animation

By

Mark William Tappan

B.S. In Computer Science, June 1982, Western Washington University

A Thesis Submitted To

The Faculty Of

The School Of Engineering And Applied Science

Of The George Washington University

In Partial Satisfaction For The Degree Of

Master Of Science

September 2002

Thesis Directed by

James Hahn

Associate Professor of Computer Science

Abstract

This thesis was driven by an interest in developing a reusable framework for building behavioral animations. The initial goal was to facilitate and automate the construction of a core behavioral animation application that could then be extended with exquisite services to investigate concepts in specific research areas. We wanted to develop a framework that would instantiate an executable behavioral animation that provided the functionality common to typical behavioral animation applications. To achieve this goal, we needed to develop a systematic approach to engineering object-oriented frameworks and then apply that approach to the behavioral animation domain.

This thesis describes the Software Productivity Consortium's (Consortium) Approach to Framework Engineering and its application to the initial architecture design of an object-oriented framework for behavioral animation applications. I developed the Consortium's Approach to Framework Engineering using best practices from multiple engineering and management processes. This approach is the initial iteration towards a full methodology for the systematic development of object-oriented frameworks. During this thesis we wanted to validate the several of the engineering activities and develop a greater insight into current behavioral animation research programs.

Using the Consortium's Approach to Framework Engineering approach, we identified several major research programs relevant to behavioral animation. We performed a domain analysis of these programs to identify major business use cases, called summary use cases, and associated actors. We combined and consolidated the summary use cases as a set of requirements for the targeted behavioral animation framework. We used the summary use cases and actors to drive the identification of system level use cases for each research program and combined these use cases into a consolidated set of system requirements. Using the actors and system use cases we developed an initial architecture containing both invariant internal services (receptacles) and modifiable public services (hot spots). We conclude this thesis with observations and summary of achievements that document our findings and experiences. Our conclusions focus on the domain analysis and consolidation of use cases between five different programs concentrated in areas relevant to behavioral animation but considerably different in focus.

Copyright Notices.[1]

[1]Company names, product names, and technologies are trademarks or registered trademarks of their respective organizations.

Dedication

For my Family, Patsy, Trina, Erick, and Alex

Acknowledgements

I owe a great debt of gratitude to my friends and colleagues at the Software Productivity Consortium, who have encouraged, challenged, confused, enraged me, and forced me to think and better myself. Most prominent include, and I apologize to those not listed for I am running short on words,

- Rich McCabe, who has consistently forced me to think through my ideas, to validate them against real scenarios, and who has inspired me to complete this thesis. I have spent many hours in long, detailed and complex conversations with Rich, and the results of those talks have guided CAFÉ.
- Lisa Finneran who has given me the latitude to explore and develop CAFÉ, and who has consistently believed in me and encouraged me, and given me the opportunity to better myself in many ways.
- The 1000+ students of the Software Productivity Consortium's Introduction to Architecture Frameworks course whom I have instructed and who have patiently listened, (hopefully) learned and have forced me to clarify my own understanding of frameworks, object-oriented technologies, and software architecture.
- Assad Moini with whom I have had many deep conversations regarding the nature of computing, the computing industry, and technology. I have (slowly) learned much from Assad and he has helped me clarify and deepen my understanding of architecture and frameworks.
- John Blyskal with whom I have had many brain-busting discussions on the nature of enterprise architectures, software architecture, and the use of standards.

My wife and best friend Patsy who had repeatedly refused to let me quit after I took all the fun classes. She has been amazingly supportive throughout the whole journey, which has included the birth of our second son, the marriage of our daughter, and many others of life's events.

My advisor James Hahn who has been patient and accommodating, even though my thesis has undergone several major focal points as I have wondered somewhat aimlessly. He has given me the latitude to try and match demanding professional responsibilities with the rigors of completing a master's thesis.

Table of Contents

Table of Figures

Table of Tables

Glossary

Application frameworks	A system design and associated code modules that enable the construction of complete applications within a domain.
Architecture	The structure, components, interface specifications, operational, system and technical requirements of a computer system.
Architectural framework	A description of the components, interfaces, standards, and requirements of a system.
Black box frameworks	Frameworks in which the developer cannot review design and implementation details of the framework.
Behavioral animation	Computer animation where main characters are controlled by modeling their behaviors resulting in emergent behavior.
CAFÉ	Consortium Approach to Framework Engineering
COM	(Microsoft) Component Object Model
Component frameworks	The design and associated runtime components for construction of component-based applications.
Connection	The link between a plug and an outlet.
Consortium	Software Productivity Consortium (http://www.software.org/)
COTS	Commercial-off-the-shelf
Creature	An autonomous character in a behavioral animation.
Ethology	The study of animal behavior.
Framework	A basic conceptual structure (as of ideas). [Webster 2001]
Hot spot	The areas where the framework can be extended or tailored to meet the needs of specific applications.
IBM	International Business Machines

J2EE	Java 2 Enterprise Edition
MFC	Microsoft Foundation Classes
Object-Oriented frameworks	A design and associated set of code modules for the partial construction of an object-oriented software application.
OMG	Object Management Group (http://www.omg.org/)
OO	Object-oriented
OOPSLA	Object-Oriented Programming, Systems, Languages, and Applications
Outlet	Services provided by any receptacle.
Plug	The mechanism inserted into the hot spot of a receptacle.
Product-line frameworks	A design for building an application from a family of applications in a single domain with known commonalties and variations
Receptacle	The internal services implemented as private classes, which are the immutable logic for the framework.
UML	Unified Modeling Language
Use case	Description of a requirement for a computer system. Summary use cases describe the business requirements or scenarios for the system. System use cases are a detailed description of the user requirements.
White-box frameworks	Frameworks in which the design and implementation are available to the application developer.

1 INTRODUCTION

A framework is simply a basic conceptual structure for organizing ideas, information, data, or anything else for that matter. Beams and rafters are elements of the framework of a house; terms, definitions, and classification schema are elements of a biology framework; components, interface definitions, and standards are elements of (software) architectural frameworks; roles, responsibilities, activities, and entrance/exit criteria are elements of process frameworks; and there are many other examples. There are enough examples and most people seem to feel comfortable with the notion of a framework, though few can provide a useful, working definition.

This thesis, and the related Consortium Approach to Framework Engineering (CAFÉ) report attempt to define a framework, an approach to building frameworks, and the application of that approach towards software development. We examine the application of frameworks to the computer graphics domain, and in particular how CAFÉ can be used to design a more generalized behavioral animation system.

1.1 Problem Statement

Effective software reuse has been a dream of corporate America for many years. Some researchers and technologists believe that this dream has yet to be realized, while others believe that it is commonplace within industry. The major point in this debate appears to be what is considered effective reuse. Operating system libraries have been used for several decades, and are arguably the primal form of reuse. Software reuse based on component models, such as Microsoft's Component Object Model (COM), has achieved another level of reuse and has fostered a minor component industry. Commercial frameworks, such as the Microsoft Foundation Classes and IBM's San Francisco Framework, provide general computing support beyond typical operating system libraries. These frameworks are considered infrastructure or horizontal frameworks since they provide services applicable across many business domains and provide a general foundation for encoding more specific business logic. Horizontal frameworks are quite complicated and powerful – they encapsulate the best brightest practices from expert software developers, architects, and designers. However, they are domain independent by nature – Microsoft Foundation Classes capture the best practices for creating Windows desktop applications, and IBM San Francisco captures the best

practices for creating Java-based business applications. These frameworks do not provide the typical business logic common to applications within a specific domain.

Problem 1. For effective software reuse within industry, engineers require a means to identify, specify, structure, and develop common domain-specific services within their industry.

However, identifying the most effective common services and defining an optimal structure for those services is a complex and difficult task. Effective reuse libraries are often costly endeavors requiring several design and implementation iterations before they are of value to developers. The difficulty lies in knowing, often as a result of significant experience, how to constrain the engineering of application solutions without compromising an engineer's ability to meet systems requirements.

Problem 2. Optimal solutions are elusive and require iterative approaches that capture and leverage the experiences of domain and application development experts.

The majority of applications within a particular business domain utilize common, domain-specific operations (or services). In many cases, these services are developed over a period of time, in isolation, without the benefit of overarching systems engineering guidance. Industry consortia, such as the Object Management Group (OMG), define standardized, vertical services through an open process of mutual cooperation among application vendors in particular domains. For example, the OMG is developing several vertical service specifications for the healthcare industry. Currently existing specifications address the need for a common Person Identifier Specification and (medical) Lexicon Query Specification. To develop these and other specifications, OMG elicited comments on proposed set of data types, data structures, and interface specifications. These design artifacts are the results of cooperation among expert application designers, architects, and developers in the healthcare domain. However, the OMG process can be quite lengthy and often takes several years before a specification is published and even more before the implementations are commercially available. Most industry efforts to build frameworks for internal use cannot afford to wait years for a viable framework, and most are reluctant to share their intellectual property with their potential competitors.

Problem 3. Organizations building frameworks for internal use need a systematic engineering process for identifying common services and architecting frameworks within their domains.

Technology change, whether through change or extinction, is commonplace. Any substantial application or system deployed into service for more than a couple of years faces challenges with technology refresh and technology insertion. Object-oriented frameworks are no different. The underlying technology, whether it is operating system, infrastructure framework, or distributed computing software, will eventually change even for frameworks. Few organizations have a well-considered approach to technology insertion and refresh for long-term applications or systems.

Problem 4. To keep a framework current and viable, the development process must address activities for identifying new technologies, prioritizing updates, and inserting new technologies into the framework.

At a recent workshop (OOPSLA 2000) on object-oriented framework construction, practitioners reported that one of the most significant issues was that application developers were unsatisfied with available frameworks. Developers complained that frameworks developed in their behalf were difficult to understand and did not meet their needs. After some analysis, most developers were incorrectly applying frameworks or were attempting to extend them beyond their intended scope. Additional analysis revealed poor and inadequate document including little or no information on the design considerations, constraints or context that drove the development of the framework. Similarly, the existing framework document focused on syntax and semantic issues and little attention was given to documenting how to apply the framework in general.

Problem 5. To efficiently and effectively use a framework, developers need detailed knowledge and understanding of the engineering design context underlying the framework. This context is critical to understanding the assumptions and constraints for the framework. Without this comprehension, developers are more likely to incorrectly use the framework.

1.2 Motivation

The behavioral animation segment of computer animation has developed a great number of impressive techniques, models, algorithms and systems to aid in the construction of believable animation. Researchers focus in specific interest areas, such as physiological modeling, cognitive modeling, learning, facial and body expressions, learning, memory, and emergent behaviors. While major contributions have been made in these and other areas, the behavioral animation community lacks a unifying model or framework to integrate these techniques together.

This thesis has admittedly had a colored and varied past. We originally started out exploring the use of distributed object technologies as a means to enable the distribution across computers of the actors participating in a behavioral animation sequence. In parallel, an unrelated effort at the Software Productivity Consortium (Consortium) had started on understanding architectural and object-oriented frameworks, and their use in software construction. As we started to develop distributed behavioral animation actors, we developed an interest to add the best features from these major animation research projects into our own animation. Meanwhile, at the Consortium a project focused on creating a method for constructing domain-specific services into an OO framework had started. The resulting merger is this thesis, the adaptation of CAFÉ towards architecting common services for behavioral animation applications.

The end goal, which is far beyond the scope of this thesis, is to construct an object-oriented framework for developing distributable, autonomous actors (i.e., agents) to participate in behavioral animation. The framework would create a generalized actor which the necessary infrastructure and communication capabilities to participate as a general actor. The framework also provides the design points for extending the actor to include and explore new capabilities, such as a new memory model. For example, suppose an animation includes 1000 actors distributed on a network. The framework provides a generic actor with core capabilities of communication, sensory input, mental facilities, etc. Developers extend some of the generic actor with unique or advanced features by extending the framework in specific ways. Almost any business domain can benefit from using object-oriented frameworks. However, those of most interest to the author include biomedical simulation, entertainment, imagery and geo-spatial management systems

The immediate goal; however, is to discern and develop a method for creating frameworks, exercise and verify the method, and apply the method towards architecting a framework based on a decomposition of several leading behavioral animation research projects.

1.3 Related Work

Object-oriented frameworks are not new inventions and are more common within the software development community than many engineers and managers realize. Behavioral animation systems are not commonplace either, but nor are they unfamiliar. In fact, behavioral animations have contributed to several major motion pictures including Batman and The Lion King. This section reviews recent work in both frameworks and behavioral animation systems. The former work being directly relevant and contributing to the development of CAFÉ and the later work is the foundation for the design of the behavioral animation in Section 3.

1.3.1 Frameworks in General

What exactly are frameworks? What are their virtues and what are their limitations? How are they applied in software development and where do frameworks come from? These are just a sample of the many valid questions regarding any technology and frameworks are no different. Let's start with the first and try to explain what are frameworks.

Earlier in this section, we defined a framework as "simply a basic conceptual structure for organizing ideas, information, data, or anything else for that matter" and gave a couple of examples. This simple, abstract definition is not very satisfying to engineers since it is rather loose and leaves much up to the reader to interpret. Unfortunately, there is no universally accepted, specific definition of "framework". We can, however, apply this simple definition to various software engineering concepts and end up defining various contexts for different types of framework. The following non-exhaustive list describes some of the more common types of frameworks.

- **Architectural frameworks** provide "guidance on describing architectures. An architecture description is a representation, at some current or future point in time, of a defined 'domain' in terms of its component parts, what those parts do, how they relate to each other, and the rules and constraints under which the parts function" (Department of Defense 1997).

- **Product-line frameworks** provide designs for building an application from a family of applications in a single domain with known commonalties and variations. Product-line frameworks are distinguished by an explicit identification and engineering of the variances between new applications that the framework can be used to develop. The scope is generally a single product within a business that has several different models. For instance, general market accounting software might be specialized for three levels: enterprise edition, small business, and personal home edition.

- **Application frameworks** provide designs and code modules for building complete applications within a domain. This means that they are constrained from both above and below. From above, an architectural framework may restrict the context in which an application must run. From below, the target development language will restrict the development environments to which the framework can be applied. The scope typically provides 80% of the design and code for building a single application. Naturally, if an organization has designed a product line framework, then the application framework can be tailored for supporting a series of applications within a single product line.

- **Object-Oriented frameworks** provide a partial design of an object-oriented software application. The intent is to support those organizations already working in or migrating to the object-oriented paradigm. The scope might be a single application domain, which could limit the framework to supporting only part of an application, such as the user interface or database access. Alternatively, the scope also might support building entire applications within a single business domain.

- **Component frameworks** provide the designs and runtime components for component-based development, typically supporting an object-oriented or object-based paradigm. Like object-oriented frameworks, component frameworks can support either a single application domain or an entire business domain. The distinguishing feature is that component frameworks are built to support application development within a particular component model, such as J2EE or Microsoft COM.

Frameworks can also be classified according the amount of design and implementation detail available to developers. These characteristics can also be combined with the above types, for example, you can have white-box component frameworks that provide application developers with details about the components and their public and private interfaces. Some combinations don't make sense, such as a black-box architecture framework, which would hide all the details of how to build architectures in a particular domain[2]. Naturally, there are numerous shades of gray-box frameworks.

- **Black-box frameworks** are frameworks in which the developer cannot review design and implementation details of the framework. The application developer relies on the documentation and details about publicly available interfaces.
- **White-box frameworks** are frameworks in which the design and implementation are available to the application developer. The application uses the interface documentation and details but also can examine the internals of the framework for clarification and verification about how the framework operates.

White-box frameworks appeal to application developers and engineers familiar with the domain and software development tools. The framework itself provides the most accurate documentation on how it is constructed and how it can best be tailored for specific applications. Black-box frameworks rely on documentation, tutorials, and example applications to communicate the purpose, context, and use of the framework.

Most organizations develop white- or gray-box frameworks for the following reasons. It is quicker and less expensive to develop white-box framework. Iterative releases of a white box framework successively refine and mature the framework to appoint where is it practical and efficient to create a black box version. The target domain for a framework also fluctuates leaving organizations reluctant to spend precious development time and dollars on short-lived comprehensive solutions.

[2] This is counter the main purpose of the architecture framework – to be a tool for guiding the development of domain architectures.

1.3.2 Object-oriented Frameworks

An object-oriented framework is a general skeleton application that conforms to object-oriented theory and provides the services common to applications in a particular domain. This general skeleton application is not intended to be a useful, standalone application. An object-oriented framework must be extended to provide the specific functionality required in a particular application within that domain.

The object-oriented framework establishes the central control of execution, and the framework invokes objects that extend the framework. That is, the framework provides mechanisms by which the objects provided by the developer will be invoked. Applications developed using this type of framework require the use of object-oriented techniques to extend the framework and use its services. Figure 1 shows that an object-oriented framework can include any or all of the following:

- Abstract class interfaces that require developers to provide class implementations
- Abstract class interfaces with reference class implementation that developers can override
- Public-class interfaces and implementations
- Private-class implementations
- Object-Oriented Frameworks and Applications

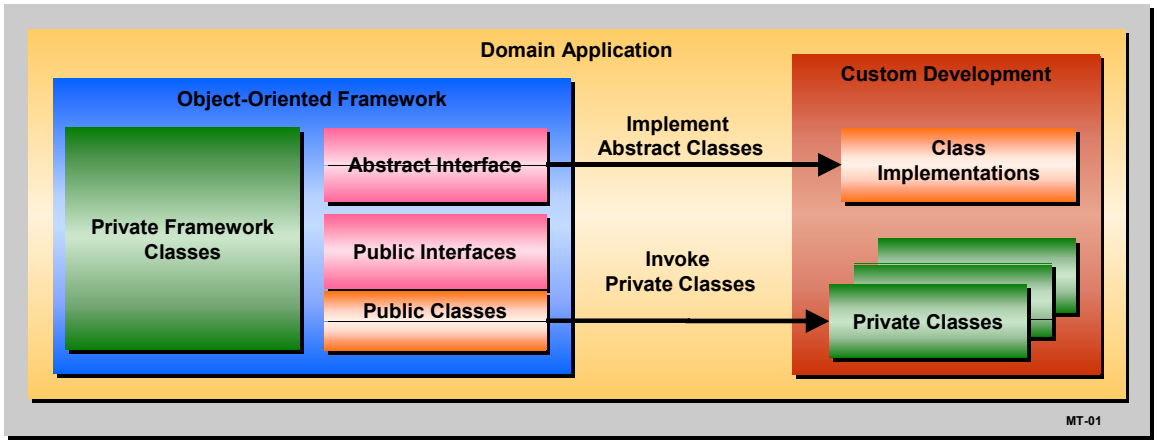

Figure 1. Object-Oriented frameworks

A skeletal application created using an object-oriented framework is an executable application that provides some level of functionality. Without further extension or adaptation, the object-oriented framework

provides whatever default behaviors were designed for it. Following are some implications that underlie this definition:

- An object-oriented framework provides the main application executive or control loop that invokes methods, including the adapted or extended methods provided by the application developer.
- An object-oriented framework provides public interfaces that the application developer can use to extend or adapt part of the framework. Object-oriented framework includes internal (private) classes and methods that cannot be adapted or extended by the application developer. Application developers only can customize object-oriented frameworks by extending or adapting public interfaces.
- Framework developers have identified and analyzed the requirements of a representative set of current and future applications to determine the private and public services that will be provided by the framework.

The following sections describe major architecture elements or themes related to object-oriented frameworks.

1.3.2.1 Anatomy of a Framework

A framework addresses the common aspects of a specific problem while providing mechanisms to support variations between different applications. Different parts of the framework express different needs.

- There must be one part that holds and organizes the common aspects of a domain.
- Another part manages the differences or variations so that the framework can be tailored for each separate implementation.
- Another advertises the aspects that are open to tailoring.

When application is made of the framework, there must be an element that removes the variation to produce a specific solution to a problem. Somehow the various framework parts must connect to each other. Finally, an interaction pattern describes how all the parts work together. An object-oriented

framework consists of dynamic parts that encapsulate the flexibility, areas that are modifiable by application developers, and static parts that are immutable and serve as the foundation of the framework. Framework literature refers to these as hot spots and receptacles, which are defined as follows:

- **Hot Spot.** A hot spot locates an area of variability within a domain and consequently within a framework. Multiple applications in any domain will have differences. When these differences stem from the same area, that area is a hot spot and should be made very flexible. Each hot spot is associated with a responsibility that must be satisfied by an aggregation of framework elements (Pree 1995; Schmidt 1997).

  A hot spot in a framework represents the known or anticipated variations in requirements for a particular service between applications in the domain. Hot spots identify areas where the framework can be extended or tailored to meet the needs of specific applications. A hot spot can be specified as an abstract class with a reference implementation of the class being provided by the framework. Similarly, it can be a public class interface (and implementation) and provide some guidance on how to extend or override the class. Hot spots can be generalized classes that the application developer specialized through inheritance. Application developers also can extend the framework by creating composite classes using framework hot spots and their own custom classes. Hot spots also refer to the customization of a framework service through polymorphism.

- **Receptacle.** A framework receptacle holds the common aspects of a problem domain. This means that the receptacle holds the data and services that must be part of every application made using the framework. Data and services are packaged according to the type of framework—from specification modules to abstract classes and components.

  Receptacles refer to the internal services implemented as private classes, which are the immutable logic for the framework. In a black-box framework, these services are completely transparent and hidden from the application developer. In white-box frameworks, these services can be discovered but are not engineered for extension or modification by the application developer.

  We will not use the term "receptacle" but refers to these services as private classes or internal services.

The following are additional terms that the reader is likely to encounter while reading about object-oriented frameworks and frameworks in general (we do not use these terms specifically, however, the concept of patterns as a means to think about design solutions):

- **Outlet.** An outlet (electrical or software interface) advertises the services of any receptacle. Tailorable services are indicated to provide a correct specification to users of the receptacle. An outlet stands ready to receive and forward incoming events. An outlet can advertise a local or remote service (Wang, Ungar, and Klawitter 1999).
- **Plug.** A plug is a mechanism inserted into the hot spot of a receptacle. The plug tailors the framework services for a particular application. This linkage is implemented by a connection.
- **Connection.** A connection links the plug into a receptacle hot spot. A connection is implemented in one of two ways. If the hot spot is a black box, then the plug must connect by selecting from the services provided by an outlet. If the hot spot is a white box, then the plug must connect directly to the receptacle's functionality at software compile time.
- **Patterns of structure.** Variability within a framework and its associated hot spots can be structured by design patterns. When several hot spots center on a similar group of elements, that group and their relationships can be abstracted into a structural design pattern used in each hot spot.
- **Patterns of behavior.** An application's behavior is constrained by the framework. The template services in receptacles define a protocol of interaction between the framework elements and the application-specific elements. Once again, when there is a recurrence of behavior among hot spots, this pattern of interaction might be abstracted and implemented as a behavioral design pattern.

1.3.2.2 Architectural Implications for Applications

An object-oriented framework imposes constraints on any application that uses it. An application framework typically provides executable code for as much as 80% of a complete application. The extent of

this influence ranges from architectural issues to application packaging, distribution, or allocation and patterns of interaction.

To use the framework effectively, the application developer must understand the basic system requirements the framework is meant to resolve and the dynamic operation of the framework. The framework's requirements description enables the application developer to understand the framework's solution space and determine whether and how the framework can be used to construct applications. The framework dynamics describe the stimuli that drive the framework and how the framework responds to those stimuli. The application developer must understand these dynamics to understand how custom objects and methods will be invoked.

The default application generated from an object-oriented framework is an executable program, and there is a sense of the framework being "in charge" —responsible for instantiating objects and invoking public and private methods. Framework developers may use a variety of object-oriented mechanisms to accomplish this control.

1.3.2.3 Frameworks and Classes

Application developers adapt and extend object-oriented framework by extending, overriding, or composing new classes with public classes in the framework. We use the following definitions and interpretations to describe these mechanisms.

> "An abstract class is a class that cannot be directly instantiated" (Booch, Rumbaugh, and Jacobson 1999, 457).
>
> Szyperski elaborates on abstract classes:"…that is no object can be a direct instance of an abstract class. An abstract class can have unimplemented methods (abstract methods). Non-abstract classes inheriting from an abstract class have to implement all such abstract methods" (Szyperski 1998, 366).

The public interfaces provided by an object-oriented framework can be specified as abstract classes, and the implementations left for the application developer or for future efforts. This approach allows the application to implement particular algorithms that conform to established interfaces. It also allows framework developers to specify interfaces and develop implementations incrementally. This approach

enables the priority interfaces to be developed and deployed without waiting for a complete implementation of the framework. The abstract classes within a framework must be extended and completed, with concrete classes.

> "A concrete class is a class that can be directly instantiated" (Booch, Rumbaugh, and Jacobson 1999, 460).
>
> Szyperski elaborates on concrete classes:"…a static description specifying the state and behavior shared by all objects that are instances of that class" (Szyperski 1998, 368).

Object-oriented frameworks provide concrete classes that implement particular algorithms for the framework's public interfaces. These algorithms may encapsulate an organization's proprietary algorithms or generally agreed-upon solutions, or they may default to implementations of an interface specification. Concrete classes can be used "off the rack," extended through inheritance or composition, or overridden by the application developer.

When several real-world objects share common properties, they can be defined by a concrete class. Because a concrete class can be instantiated, it is used to implement the functionality sketched out in an abstract class. When it is instantiated, then the functionality can be realized. Concrete classes are instantiated into objects that are runtime entities, which do the work of an application and provide the capability of the system (and the framework) to system users.

> An object is "an entity with a well-defined boundary and identity that encapsulates state and behavior; an instance of a class" (Booch, Rumbaugh, and Jacobson 1999, 464).
>
> Szyperski states that an object is"… a unique identity, that is can be consistently distinguished from all other objects of overlapping lifetime and access domain, irrespective of changes to its or other objects' state" (Szyperski 1998, 376).

Object-oriented frameworks provide access to instantiated objects with particular capabilities that can be useful to the custom application. Objects are available to the application at runtime.

1.3.2.4 Hot Spots in Object-Oriented Frameworks

Our definition describes object-oriented frameworks as partly immutable services and partly mutable services. The term hot spots refers to the identification of public classes in a framework in which customization is appropriate and expected. The list of hot spots includes elements of a framework in which the framework developers expect application developers to provide custom software.

The immutable services of a framework represent the common portions of domain applications that remain constant from application to application. Framework developers analyze the requirements for typical applications in the domain and identify common services that are stable and those likely to vary between applications. The stable services are identified and implemented as the core of the framework. Likewise, services that are common but likely to vary between applications are implemented as the mutable services of the framework.

The term "hot spot" refers to the public classes in a framework in which adaptation and extension is appropriate and expected. We call the classes that represent the mutable services of an object-oriented framework.

1.3.3 Behavioral Animation Projects[3]

The primary experiment we investigated was the design of an object-oriented framework for creating behavioral animation applications. Behavioral animation is an active research topic in several academic field including computer graphics, artificial intelligence, and robotics. As a foundation for the design work described in Section 3, we studied nine major behavioral animation programs from primarily the computer graphics field. There are obvious areas within the framework where additional contributions could be used to further the number and quality of frameworks services. For example, the artificial intelligence community has studied and developed several models for learning which could be adapted into the framework. Likewise, the framework could be expanded to include sensor and vision concepts from the robotics field. Table 1 lists the research programs, which are further described in the following sections.

[3] The research project described here are the results of tremendous effort, diligence, and thought. It is hardly fair to summarize them in a few paragraphs. The contributions made by each effort are far more significant than summarized here.

Table 1. Inventory of Behavioral Animation Research Projects

Behavior Animation Project	**Principal Investigator**	**Contribution**
"Flocks, Herds, and Schools: A Distributed Behavior Model."	Reynolds, Craig, [Reynolds 1987]	Basic distributed animation architecture of sensory perception, behavioral rules and selection, and motor skills.
Old Tricks, New Dogs: Ethology and Interactive Creatures	Blumberg, Bruce, Massachusetts Institute of Technology. [Blumberg 1997]	Ethological and classical animation based approach to developing "lifelike" autonomous creatures.
Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior	Tu, Xiaoyuan, University of Toronto. [Tu 1996]	Focus on realistic appearance, motion, and behavioral of autonomous creatures.
Making Them Behave. Cognitive Models for Computer Animation	Funge, David, University of Toronto. [Funge 1998]	Adds formal semantics to the specification of high-level behaviors and actions using Situation Calculus.
Lifelike Computer Characters: the Persona project at Microsoft Research	Ball, Gene (et al), Microsoft Research. [Ball c. 1996]	Improved communications, i.e., understanding of spoken phrases and selection of oral responses, between autonomous creatures and interactive users.
Modeling Emotional State and Personality for Conversational Agents	Breese, Jack, Gene Ball, Microsoft Research. [Breese, 1998].	Insight into how emotions affect the decision process and motor controls.

The Distributed Behavior Model is a central theme of the behavior animation framework we design in 3. Each of the research areas that we investigated contributed unique aspects of behavior animation to the framework. However, the real genesis of our framework comes from a simple programming assignment in a Computer Animation course at George Washington University. The assignment was to create a flocking model that demonstrated emergent behavior amongst autonomous actors. This assignment was based on Craig Reynolds' seminal paper on animation, *Flocks, Herds, and Schools: A Distributed Behavior Model* [Reynolds 1987].

My simple model required that each "boid" stay within a certain distance of its neighbor, not too close and not too far. Each boid tried to maximize its survival chances by migrating towards the center of the flock and by avoiding contact with non-boid objects. My boids could not stop nor could they travel in reverse, and they had to follow a boid designated as the "lead boid." I programmed the lead boid to fly in a circular pattern at a constant speed. Using a flock with just a few boids, I found that the flock did stay in task and did follow the lead boid around the circle. I also found, to my dismay, that as I added more boids, the flock grew in length until it was shorter for some boids to cross the circle rather than follow in line. In effect, several boids were cheating by cutting across the circle and jumping to the head of the flock. After some analysis, it was apparent that my boid possessed a rather rudimentary decision model and a simple model that allocated energy to actions. In addition, I had made no attempt to model any environmental elements such as air thermals, wind, friction or hostile creatures.

From this simple experiment, Reynolds' paper, and investigations of the projects listed below, a generalized architecture became apparent. Each of the research projected investigated provides a unique contribution to behavior animation, and helps to specify a unique capability of the behavior animation framework. The set of projects we investigated is by no means exhaustive, additional projects from the computer animation domain or from completely different domains (e.g., robotics and psychology) could be included and used to extend the framework. For example, research programs in robotics could be analyzed and adopted to provide a route planning service. Table 1 summarizes the insights and contributions from the research projects we investigated.

1.3.3.1 Flocks, Herds, and Schools: A Distributed Behavior Model

Craig Reynolds describes a model where animal actors such as birds and fish dynamically and autonomously control the action of their own animation. They are guided by behavior rules that mimic the values and constraints of their real world counterparts. Rather than key frame individual movements or calculate the kinematics for each actor, Reynolds' Distributed Behavior Model creates emergent behavior amongst its actors.

1.3.3.2 Old Tricks, New Dogs: Ethology and Interactive Creatures

The Ethology-based behavioral animation project focuses on the development of Silas an animated dog. The *Old Tricks, New Dogs* doctoral effort investigates how the study of animal behavior can be the basis of interactive agents. Principal concepts from his research include the hierarchical specification of behaviors, grouping of behaviors, mechanisms for enabling and enacting behaviors, the modeling of Silas' motor system and action selection. Blumberg's research also investigated the modeling of sensor input, such as vision, into the behavior and action selection.

Blumberg investigated the construction of autonomous creatures for the ALIVE project at the Massachusetts Institute of Technology, specifically a creature named Silas T. Dog. The basic architecture of these creatures is a multi-layered approach consisting of Geometry, Motor Skill, and Behavior. These layers are depicted in Figure 2. The Sensory Input element models and simulates various sensory inputs such as sight and hearing. The Sensory Input data is a primary driver of the behavior identification and selection system. Different behavior selections, such as chase or lay down, utilize different computational models in the Motor System to accomplish the desired behavior. It is the responsibility of the Motor System to drive the Geometry System to realize the animation sequence of motion.

The Behavior System is a major contribution to the proposed animation framework. It provides a competitive environment where potentially viable behaviors compete for the highest priority when issuing commands to the Motor System. The Behavior System models a "release mechanism" for behaviors. These mechanisms model events of objects that enable the potential selection of particular behaviors. For example, a swooping bat might be a releasing mechanism for a panic behavior. These mechanisms filter out

inappropriate responses to situations and allow a degree of control of the selection of behavior. For example, a mechanism might vary the situations where a creature feels hungry.

A Releasing Mechanism creates a structure called a pronome, which enables a sort of reuse of simple behaviors in a variety of situations. Pronomes might model jumping behavior for fish that could be enacted as part of a fight-or-flee behavior or as part of a courtship ritual. Pronomes are important for the framework as a mechanism for reusing or multi-purposing behaviors and enabling the construction of complex behaviors from rudimentary ones.

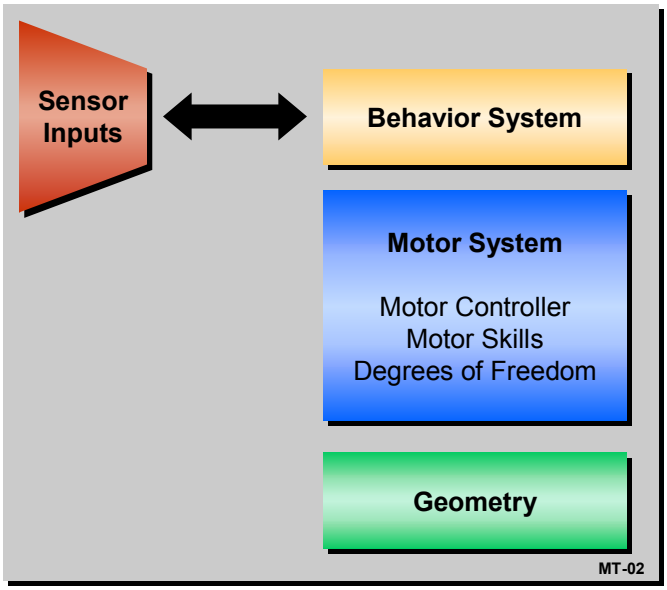

Figure 2. Basic Creature Architecture for ALIVE

Blumberg experimented with a number of other dimensions for creatures in ALIVE including behavior specification, behavior adaptation, learning, short-term memory, sensory input modeling. These elements of distributed behavior animation are important elements of in ALIVE, and equally important to the success of our framework. Adoption of these techniques would certainly be applicable to our framework; however, we are also interested in integrating disparate elements from other research projects.

1.3.3.3 Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior

Natural and realistic motion is important and necessary to the development of lifelike and believable autonomous creatures. Whether we are animating an autonomous creature as part of a film or creating a

believable computer assistant, natural motion is a high-priority requirement. Xiaoyuan Tu has made exemplary contributions to behavioral animation through the Artificial Animals project at the University of Toronto. The goal of this research program colloquially referred to, as Fishes, is to create realistic motion and appearance of autonomous creatures, as they exist in animation.

The architecture of Tu's Fishes is segmented into three major subsystems: a brain model, a biomechanical or motion model, and a graphical display model. The brain model manages the collection of sensory input data, the enactment of the behavioral model including habits, intentions, and behaviors, and the resulting motor controller commands. The motion model receives input from the brain model and executes the physical motion as directed by the motor commands. The graphical display is responsible for rendering the fish. Figure 3 shows the major elements of the architecture.

The Brain Model is responsible for sensing conditions and events in the environment and enacting appropriate action to meet the creature's goals, such as feeding. It consists of a Perception Model that is a combination of sensor perception and information filtering and interpretation. Perception modeling is a core capability exhibited in many different behavioral animations systems, and has obvious links to deeper research into computer vision, artificial intelligence, psychology, cognition, and other fields. The architecture allows for vision sensors and extra-sensory perception sensors such as temperature sensing. Filtering is used to constrain the information collected through sensors and passed to the decision-making element, the behavior model. The behavior model combines creature goals, such as eating, fighting, and fleeing, with its habits, such as preference of warmth over cold, and its behavioral rules. These rules are encoded into a hierarchical flow that prioritizes basic survival action over other concerns. This rule modeling derives from ethology and is consistent with the philosophy underlining Blumberg's research on the Alive project.[4] Using sensory inputs and these behavior parameters, a series of lower level commands are generated to move the creature.

The Biomechanical Model is responsible for interpreting the commands from the Brain Model and converting them into elementary motor commands. There are a number of approaches to modeling the biomechanical or motion model of a creature. Researchers like Jane Wilhelms (University of California,

Santa Cruz) have experimented with developing physiological models of animals, which yield remarkably accurate and realistic motion. Tu's Fishes employ a less sophisticated, but computable and highly realistic model based on springs and dampers. Creatures are modeled with a set of node points (dampers) and arcs (springs). By varying the stiffness of the springs and resistance of the dampers, different motion effects can be modeled. Tu's uses the elastic properties to model the muscle expansion and contraction of fish muscles. By contracting and expanding the springs, different "muscle movements such as swimming are possible.

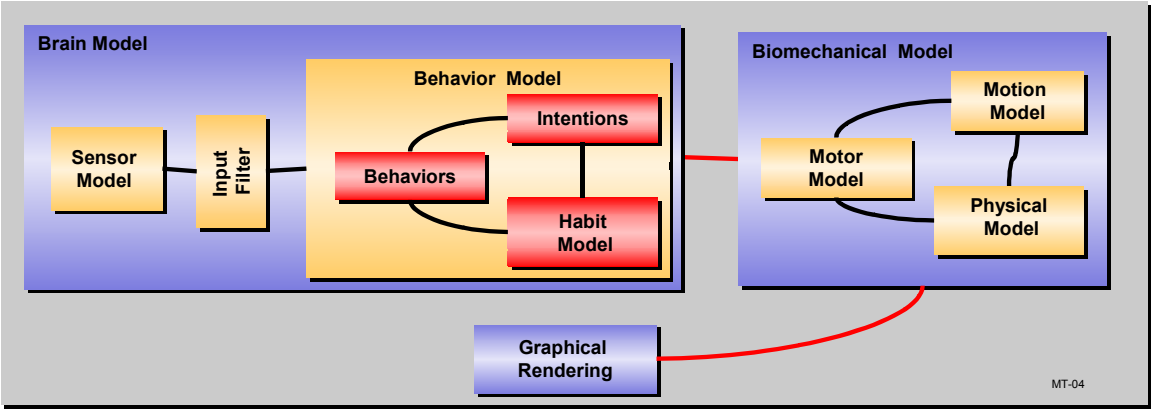

Figure 3 The Architecture of Tu's Fish.

While we did not directly adopt the environmental modeling, it does serve to emphasize that creatures exist within a physical world[5]. The creature's physical environment model has a profound effect on the realistic motion (e.g., gravity and resistance) and sensor input (e.g., foggy nights or strange smells). Virtual world, or real world modeling is an important aspect of realistic animation, and we note in Section 4 that there is an opportunity for further work.

1.3.3.4 Making Them Behave. Cognitive Models for Computer Animation

John Funge studied approaches to constructing controllers for high-level behaviors of creatures in behavioral animation. He distinguished low-level behaviors as common among many creatures, for example obstacle avoidance. He defined high-level behaviors as unique to particular species, for example chimpanzees "fishing" for termites. He proposed that the representation of a creature's knowledge is essential in developing a cognitive model for the creature. The cognitive model, interpreted by either

[4] The behavior modeling developed and emphasized, as part of the Alive project is more detailed and expressive than used in the Fishes architecture.

human or computer, requires precise semantics of the representation to avoid ambiguities. The solution he chose for this semantic representation is a language called Situation Calculus.

Situation Calculus is a rigorously defined language that represents the world as a series of situations. The language enables the user to define assertions, relationships, possibilities, and assignment of value. Using these statements, logical statements of a situation and its resolution are possible. For example, situation calculus enables users to define that a water bottle must be opened and non-empty before someone can drink from it. Similarly, the user can define that Betty wants to drink from the water bottle and if she cannot she becomes angry.

In the research projects we studied, autonomous behavior and action are specified in some fashion, often by developing a particular language or notation. In our framework, we would like to provide users an ability to specify behaviors and actions in a non-ambiguous fashion. Situation calculus seems like a valuable contribution towards the specification of autonomous behavior. As an example, for the cheating boids in my boid animation,

> *Poss* (FlyStraight, b) ^ CutCrossCircle (b) => ActionCheat(b)
>
> "If it is possible for boid, b, to fly straight and boid, b, chooses to cut across the circle, then this action is cheating for boid, b."

It has been our experience that the keys to interoperability of systems, the reuse of class and software, and the refactoring of systems lie in the ability to state and comprehend the *syntax* and the *semantics* of interfaces. Whether or not situation calculus is expressive enough, or extensible enough, or rigorous enough, is an open question. Other formal languages may be more precise or more expressive, but tend to be very difficult to comprehend and master.

1.3.3.5 Lifelike Computer Characters: the Persona project at Microsoft Research

In animations involving multiple autonomous creatures, interactions between creatures involves defined languages and notations with specific vocabularies. For expediency sake, creatures are constructed with a

[5]Regardless of whether that world is real physical world or imaginary physical world. Fortunately, the discussion of real vs imaginary physical worlds is beyond the scope of this thesis.

predefined language specific to their tasks, environment, and goals. However, in environments where autonomous creatures must interactive with human users such limited communication skills may prove too restrictive. In broader animation and even realistic simulation environments, creatures will need to communicate with other animated creatures that have different goals, vocabularies, and communication skills. It seems reasonable to include services within our behavioral animation framework that could provide these skills or at least provide a hot spot for future communication technologies.

`The Persona Project at Microsoft Research focused on the investigation and exploration of technologies to construct highly automated, highly skilled *assistants* that support the computer users. The goal was to develop interactive, personable assistants that were more like human assistants and less like bland, sterile help files, databases, and the Frequently Asked Questions lists. While not specifically a goal of this thesis or our behavioral animation framework[6], the research was interesting and demonstrates how additional features can be integrated into a framework by investigating and including innovative concepts.

Figure 4 shows the high-level architecture used in the Persona project to create Peedy[7], a conversation agent who interfaces between human customers and a CD player. The elements process the user's spoken commands, in the form of music requests. The system responds by playing from a CDROM player and with reactive gestures from the assistant. The *Speech Recognition* element senses input from the user and create a series of events for major utterances and filters out background and other noises. This element maps user utterances into elements of a context free grammar. These grammatical elements are matched against a database of known proper names to distinguish them from normal speech. The Natural Language Processing element then analyzes the input stream to extract meaning from the grammatical elements. The Semantic Elements match the analyzed grammatical elements to appropriate actions and known objects. The Dialogue Management element constructs appropriate gestures and responses based on the state of the interaction with the user. If coupled with an emotional model, the dialogue management could generate appropriate emotional responses, such as frustration. The Speech Controller element constructs the proper speech elements for output back to the user. Finally, the Animation Control element creates the associated gestures, expressions, and other noises to aid in the personification of the assistant.

[6] It also seems to demonstrate that frameworks are not immune to "requirements creep", the perilous introduction of additional requirements mid-stream in a development cycle.

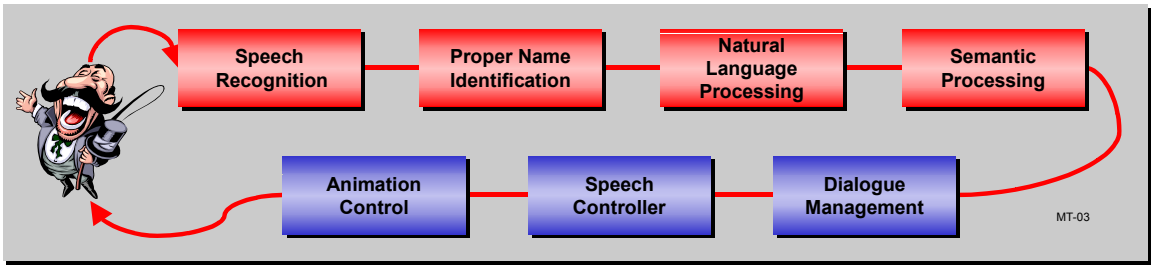

Figure 4. The High-Level Architecture of Persona Project

The Persona Project demonstrates the type of technologies and issues related to improving the interaction between humans and automated assistants or autonomous creatures. There are numerous applications of advanced interactions techniques in industry, entertainment and military applications. In entertainment for example, imagine a gaming environment, such as a networked adventure game, populated with various automated creatures and visited by human players. Creatures can expect to interact with various human users for various purposes. Today, these interactions tend towards canned and static interactions rather than allowing the player to explore deeper and broader communication. For example, in LucasArts' pirate adventure, *Return to Monkey Island*, the main character (i.e., the player) interacts with several individuals throughout the game. The player is presented anywhere from 2-5 dialogue choices; however, players often find themselves wishing to ask different questions or make different replies.[8]

1.3.3.6 Modeling Emotional State and Personality for Conversational Agents

There is more to communication than simply understanding spoken words and selecting a response appropriate to the situation. Facial gestures, body language, emotional content, and voice inflections can be as important or more important than the words. Autonomous creatures, if they are interact with humans, must identify and comprehend the emotional message associated with messages. While researching the Microsoft's Persona Project, we became interested in their related research into emotional modeling. We propose that this effort yields some interesting avenues for expanding the behavioral animation framework, and in particular could expand the quality of communication mechanisms in the framework that are based on the Persona project.

[7] Peedy is a personification of a parrot who provide an avatar interface to a music system.

The Modeling Emotion research seeks develop a systematic approach to identifying the emotional and personality basis of spoken expressions and produce an appropriate response based on the emotional and personality makeup of the agent, or creature in our case. The establishment of the emotional content of a creature produces modifiers that alter the behavior, motion, gestures, comprehension, and speech. This research effort builds a Bayesian Network that maps utterances to emotions and personality traits. These in turn are mapped into the personality and emotional Bayesian Network for the creature to produce modifiers. Modifiers are then passed to the pertinent elements, such as a behavior model.

The proposition of improving the realistic quality of autonomous creatures, their inter-relationships, and their relationships with interactive human users is very attractive. Suspecting that this research effort into modeling of emotion was not unique, we undertook a brief and cursory search into the field. We discovered that substantial effort and progress has been made in modeling emotion, and naturally several competing theories have been developed. As we will see in Section 4, the modeling of emotion in the framework is a prime area for future research. One benefit or advantage related to the framework development is that it can provide a comparative tool for evaluating similar techniques, philosophies, and approaches. We suggest that the framework could be extended to provide a test bed for evaluating various emotion models.

1.3.3.7 Summary

The initial selection of projects represents a focus on different aspects of behavioral animation and represents a core of capabilities that our framework will provide. These basic capabilities provide a simple, but comprehensive end-to-end model for behavioral animation. The core capabilities enable an animated actor to maintain first and second order objectives, identify and collect sensor inputs, apply to sensor input, decide on a course of action, and drive motor sensor functions. In some cases, such as with emotion modeling, capabilities where identified for future adoption.

1.4 Proposed Approach

Process and methodology development is an arduous and iterative task by its nature. The proper development of a process or methodology requires many design, document, and review sessions within a knowledgeable and experienced group to come to consensus on generalized approaches to solving

[8] To be fair, this type of simplistic interaction dramatically improves the gaming experience over less communicative

particular problems. It is far simpler to document the process one uses to accomplish a task for one's own purposes. However, unless you are a recognized expert in the task at hand, your process will be your own and will not benefit a larger community. Our approach was to develop an approach, without the benefit of a prolonged review process that could be adopted and used within any software development community.

Our approach was to identify basic management and development tasks and select commonly accepted practices for those tasks. Our goal was to integrate these disparate tasks into an approach, which could then be tested, refined and verified by a smaller community. Our approach to developing CAFÉ and exercising the approach is:

- Develop an initial methodology[9] for the construction of object-oriented frameworks
- Exercise and refine this methodology by the construction of a simplistic object-oriented framework
- Apply the methodology to the design of a large, comprehensive framework targeted at behavioral animation

1.4.1 Initial Methodology Development

In 1997, the Software Productivity Consortium (Consortium) held a workshop amongst its members to discover potential areas where the Consortium could help its members with (essentially) systems based heavily on commercial-off-the-shelf (COTS) software. One result from that workshop was an intense interest in the impact of architectural frameworks on their software development efforts. This ultimately led to the development of the Consortium's Survey of Architectural Frameworks and Integration Tools and the Introduction to Architecture Frameworks course. Both of these products provide background information on frameworks and their characteristics. The course generated sufficient interest in framework to warrant a task in 2000 to develop an approach that would allow organizations to develop their own frameworks. This

games, such as Sierra's blockbuster, Half-life.

[9] A brief discussion of how *approach*, *method*, and *process* are used within this thesis. A *process* is a set of activities that describe how to reach some ultimate goal. A *method* is a sequential set of steps that can be followed to achieve an immediate goal. An *approach* is a set of activities and steps under development that have not matured into either a *process* or a *method.*

product, released in 2001, is the Consortium's Approach to Framework Engineering[10] (CAFÉ). Section 2 describes CAFÉ in more detail.

1.4.2 Application in Behavioral Animation

CAFÉ was then refined and applied to the design of a framework for behavioral animation based on several major research efforts in the area. Several research projects were identified, analyzed (i.e., read), and used to identify and select behavioral animation services for the framework. This effort, in addition to developing a strong background in behavioral animation research, refined the CAFÉ approach to requirements gathering and service identification. Section 3 describes the approach and refinements of applying CAFÉ to behavioral animation..

1.5 Thesis Organization

This thesis is organized as follows:

Section 1 Introduction. This Section provides background information on the purpose and motivation guiding this thesis.

Section 2 CAFÉ – Consortium Approach to Framework Engineering . This section describes the approach for developing frameworks created by the author while at the Consortium.

Section 3 Behavioral Animation. This section describes the application of CAFÉ to the design of a comprehensive frameworks focused in behavioral animation systems.

Section 4 Future Efforts. This section proposes additional ideas and potential applications of CAFÉ and object-oriented frameworks both in and out of the computer graphics and animation domain.

1.6 Typographical Conventions

The typographical conventions and symbols used in this report are listed in Table 2.

[10] I must give credit to Rich McCabe for the name CAFÉ. My original names were CAFD (Consortium Approach to Framework Development) and CATFOOD (Consortium Approach To Framework-based Object Oriented Development).

Table 2. List of Conventions

Typography	Convention
Georgia	Conclusion drawn from this thesis
Time New Roman	Main document text

2 CAFÉ – CONSORTIUM APPROACH TO FRAMEWORK ENGINEERING

The construction of object-oriented (OO) frameworks is not a new endeavor. Common commercial examples include the Macintosh development environment, MacApp; Microsoft Foundation Classes (and supporting toolset); the X Windows System, and IBM San Francisco Development Environment. In industries such as healthcare, manufacturing and finance, OO frameworks are being developed to capture domain expertise, simplify software construction, and reduce cost and time to market. Ralph Johnson, Brian Foote and other leaders in the OO framework community have provided guidance, tutorials, and case studies. However, there is no specific software development method or process focusing on the construction of OO frameworks.

The nature of software development methods and processes being applied to the construction of OO frameworks is ad hoc –they are difficult to plan and manage, costly and time consuming, and often produce frameworks of inconsistent quality and utility. Using the goals described in 1, there is a need to develop an approach to constructing OO frameworks using a set of manageable, reliable, effective methods. The Consortium's Approach to Framework Engineering is an initial effort into developing such as approach.

2.1 Introduction

CAFÉ is a set of activities for the development of object-oriented frameworks that helps organizations define an object-oriented framework's business and technical scope, analyze requirements for target applications, and develop an effective framework architecture. The CAFÉ process provides guidelines for identifying the key services needed to develop target applications within the domain. CAFÉ is a development process that implements framework services from a prioritized set of requirements. It is anticipated that a framework development team will apply CAFÉ repeatedly to continue evolving and extending these framework services over time. This incremental nature provides the highest priority services first and then implements lower priority services in subsequent development phases. It also helps organizations manage technology change by providing opportunities for technology insertion and identifying technology obsolescence.

Object-oriented frameworks, like other software applications, are developed through a series of management and technical activities that govern the life cycle from requirements engineering to deployment and operations. There are issues beyond application development that are unique to the development of object-oriented frameworks. The following list is an indication of some of the major issues related to framework development:

- Management sponsorship is key to defining, institutionalizing, and evolving the framework.
- Object-oriented frameworks are an enterprise-level approach to reusing significant portions of target applications. The development, deployment, and maintenance of a framework requires commitment and support from the organization.
- Consensus among framework developers, target application developers, and stakeholders is key to developing and deploying a viable framework.
- While technology and subject matter experts lead the framework development team, stakeholders ensure through consensus that the framework meets business objectives and can be used by application developers to build target applications. There also must be a consensus that the common services of the framework represent the needs of the user community.
- Development of an object-oriented framework is a long-term commitment in which investment returns are realized incrementally as new framework services are deployed. This commitment includes a requisite operational and maintenance cost, which covers typical periodic costs such as defect resolution and infrastructure upgrades.
- Organizations deploying frameworks indicate that successful frameworks are the result of many successive, evolutionary development and deployment cycles. Many organizations and framework development projects report that it takes several development and deployment iterations before a framework is an effective tool supporting target application development. The major issue is identifying the common services likely to be required by future target applications. In addition, organizations are reluctant to spend up to 2 years developing a framework before using the framework in development scenarios. Incremental development and deployment releases of a framework enable organizations to use some services while others are being developed.

- Defining the most effective set of common services and hot spots requires a detailed, expert analysis of the target application domain and technology environment. An object-oriented framework provides a skeletal application with common services typical of applications in the target domain. Identifying and packaging those services requires careful and expert consideration. This analysis will help decide which services are likely to change (hot spots) and how those services might best be used from an application developer's perspective.

- Application developers need a clear understanding the intent, context, and utility of the framework in order to adopt and tailor it correctly. Misuse of frameworks is a major impediment to meeting the business and organizational goals for developing the framework. Often frameworks are documented through a description of the hot spots and overall purpose of the framework. White- box frameworks give application developers greater access to framework internals, which can lead to unintended and unsupported tailoring of the framework. Framework documentation must provide a clear description of the intended architecture, support the adaptation of common services, and reference implementations of target applications.

- Framework verification requires the development of reference applications to exercise variations provided by the framework hot spots. Framework hot spots are locations within the framework in which variation through adaptation and tailoring is expected. There is no pragmatic restriction to how these services might be adapted or tailored; verification of hot spot services under these conditions is difficult. Typical approaches to framework verification include the development of reference applications to exercise the hot- spot interfaces and tailoring mechanisms.

The following sections introduce the management and technical aspects of CAFÉ, while the activities, issues, and roles comprising those aspects are further described in the Management and Architecture sections.

2.1.1 Framework Management Practices.

The framework management practices initiate the framework development process and control the evolution and deployment of the framework throughout its development phases. They consist of the steps necessary to establish the framework team and executive sponsorship, develop information exchange

mechanisms, and define the framework management procedures (e.g., configuration management, testing, and compliance). The goal is to develop management practices necessary to ensure the successful implementation and application of the framework throughout its life cycle.

Framework management practices also define the current and anticipated business and technical and objectives for the framework. These are key elements for determining the scope of the technologies, services, and components included in the framework and for determining which technologies will be included in future development cycles. The framework scope refines and bounds the framework—predetermining some architectural choices for target applications and improving the usability of the framework. The goal is to define the mission of the framework accurately so that architects and developers can understand the context for the framework. The framework management practices are as follows (process integration is not addressed in this release of CAFÉ):

- **Identify framework team.** The framework is assigned to design, develop, maintain, and transfer the framework. A central issue related to establishing a framework team is to ensure the viability, correctness, and utility of the framework. A strong commitment by the organization's management is necessary to engage corporate commitment to developing and institutionalizing the framework in addition to evolving the framework over time. It is essential to engage the best domain and technical experts to ensure that the correct framework services are identified and properly constructed into a usable framework. Finally, users of the framework (application developers) in addition to the target application are critical to understanding how the framework and target systems are used to support business operations and objectives during application development.

- **Define stakeholders and goals**. The Identify Stakeholders activity identifies the individuals or groups who have a significant vested interest in framework, describes their roles relative to the framework, and describes their interests in the frameworks. Stakeholders represent two distinct interests in developing an object-oriented framework. The management stakeholders are concerned primarily with relative business interests, which include whether the framework will help the organization meet business objectives such as reduced time to market or lower development costs. Management stakeholders also are interested in providing control and support

for the framework development throughout its life cycle. Typical project management goals include the proper consideration and prioritization of the framework services so that highest priority services are implemented before lower priority services, allowing the framework to evolve within the cost/benefit constraints. It is critical to identify management stakeholders, as they will be the ultimate champions, benefactors, and overseers. Without management commitment and attention to management stakeholder goals, the framework will have little chance of meeting business objectives and little chance of success.

The other group, the technical stakeholders, is concerned with the design, implementation, and use of the framework. The goal for this group of stakeholders is to identify the correct set of internal services and hot spots for the framework. Proper selection of these services is of paramount consideration to ensure that application developers can create target applications that will meet the needs of the end users.

- **Collect supporting information resources.** The Supporting Information Resources activity identifies and collects information resources that may be useful to the framework team and target application developers using the framework. The objective is to develop a repository of information that can provide framework and application developers with some understanding of the context, rational, and decisions that went into developing the framework. Furthermore, the repository helps describe the technical environment, constraints, intended usage, and purpose of the framework. The repository contains background information, framework documentation, and training materials to help developers avoid an unintentional misuse of the framework.

- **Identify technology standards.** The Identify Technology Standards activity identifies and collects the standards and guidelines that establish a technical foundation for the framework. In the broad sense, framework standards include technologies, products, software assets and de facto industry and international specifications that have been adopted or mandated within the organization. These enterprise standards provide the common basis between the framework and its target applications. Like frameworks, enterprise standards are an enterprise solution to business concerns, including application consistency, application interoperability, and staff training and

skill sets. Enterprise standards are also a form of architectural constraint in that they predetermine certain engineering decisions and become anchor technologies. Within CAFÉ, anchor technologies include engineering and industry standards, commercial products, and required software assets that are mandated for use by target applications and are elements present within the framework.

- **Define compliance procedures.** The Define Compliance Procedures activity develops the rules to measure the degree to which a target application adheres to the services, standards, and guidelines provided by the framework. While the phrase "compliance rules" conjures up images of ominous review boards and stifled creativity, there are valid and important benefits behind establishing compliance rules. These rules help organizations institutionalize development practices using a framework (and other technologies, tools, and practices) to meet the business objectives driving the development of the framework. Compliance rules can help organizations determine how applications are using (or not using) the framework. The rules also serve as guidelines that help development projects follow proper development practices. The degree to which an organization dictates compliance is largely driven by the demands of the particular domain and, to some degree, a personification of the organization and its values.

- **Establish change management practices.** The Technology Change Management activity addresses the need to monitor, predict, and manage technology change relative to an organization's framework or frameworks. Technology change management requires considerable effort to identify the key technologies (including development environments, infrastructure, methodologies, and other elements supporting the framework) that are likely to change over the lifespan of the framework. Change may be driven by new versions of products, new products introduced into the market, early adoption of emerging technologies, or obsolescence of existing technology. Change can be driven by new business objectives, such as a migration toward e-commerce, anticipation of a demand for new application features, or unavoidable changes in the marketplace, such as a vendor discontinuing a product. The essence of these activities is to be aware and sensitive to changes in technologies that may require significant reengineering of the framework. Technology change requires careful planning and resource allocation to minimize the cost and schedule impact on the organization.

- **Integrate framework-based development processes**. To have a successful framework, an organization must adopt and use the framework as part of its practices for developing target applications. In order to gain the fullest possible benefit from an object-oriented framework, the development and evolution of the framework and its use as part of application development must be a normal activity in an organization's software development processes.

CAFÉ management practices are intended to augment the business and technical management practices used in organizations to development software-intensive systems. These practices address issues of specific interest to object-oriented framework developmental, though organizations may already include them in processes. Additional practices may be needed to institutionalize and improve the efficiency of framework-based development. CAFÉ management practices and their associated inputs and outputs are depicted in Table 3.

Table 3. CAFÉ Management Activities

Input to / Requires	**Management Practice**	**Produces**
	Identify Framework Team	Executive Sponsorship Subject Matter/Technical Experts Stakeholders
Executive Sponsorship Stakeholders	Identify Stakeholders and Stakeholder Goals	Business Use Cases
Subject matter/Technical Experts Business Use Cases	Collect Supporting Information Resources	Information Repository/Training
Subject matter/Technical Experts Stakeholders Business Use Cases	Identify Technology Standards	Framework Standards Enterprise Technical Architecture
Stakeholders Business Use Cases	Define Compliance Procedures	Compliance Rules

Subject matter/Technical Experts Summary and System Use Cases Framework Architecture	Establish Training Program	Training Materials
Stakeholders	Establish Technology Change Management	Technology Watch Plan Potential Technologies
Subject Matter/Technical Experts Information Repository/Training Compliance Rules	Integrate CAFÉ Into Existing Development Processes	Integrated Processes

2.1.2 Framework Architecture Practices.

The framework architecture practices capture the knowledge of subject matter experts and identify the key functionality and operations of typical and projected applications in the domain. This knowledge is augmented with evaluations of standard architectures and reference implementations from the domain. Domain analysis steps help framework architects plan for the future by anticipating services from emerging technologies. The goal is to identify a set of canonical services essential to applications within the domain and to anticipate what services will be essential to new applications in the domain. This analysis leads the framework team to understand which common services should be inherent and immutable to the framework and which services are hot spots representing variations between target applications.

The framework architecture step establishes use cases to form the system and software requirements for the framework. Use cases also can be the basis for specifying and developing reference applications using the framework. Reference applications help framework developers exercise the framework elements and structure and capture some of the context underlying the design decisions and help document the framework.

Object-oriented analysis techniques are used to organize the key services and to define the interaction patterns between the various framework elements. Abstract classes are used to define interface specifications for these framework elements. The goal is to define system and software requirements for

key services within the scoped domain and to use those requirements to define the framework structure and service representation. The following are object-oriented analysis techniques:

- **Define framework domain.** The Define Domain Definition activity establishes the scope of the services provided by the framework. This activity is used to analyze the target domain identified in by the stakeholder goals and expressed in business use cases.

  The framework must provide the services that application developers require to construct applications. If the framework service cannot be used to create target applications, then application developers will create services on their own. The identification of these "correct" services, and how application developers “correctly” use those services, requires domain and technical expertise to evaluate current application trends within the domain. This analysis can take many different forms; for example, it could consist of an extensive architectural review of existing applications, a survey of common features among users, audit logs from applications detailing users common operations use, and interviews with subject matter experts to determine common services. Furthermore, the intention driving the development of an object-oriented framework is to support future application development; hence, the domain analysis must include some level of educated evaluation of potential future trends in the domain.

- **Capture behavioral requirements for common services**. The Capture Behavioral Requirements activity initiates the framework design process by refining the summary use cases and stakeholder goals into requirements specifications. The behavioral requirements for a framework must capture and express the internal (private) services and the public (hot spot) services. These requirements can be captured in the form of software (sometimes referred to as system) use cases. Because the framework is constructed through a series of development life cycles, the software use cases must be prioritized to determine the most effective implementation order.

  The software use cases must express the services common to applications in the chosen domain. They represent the infrastructure services that developers will customize and extend to construct specific applications. Software use cases must capture the behavioral requirements, human and nonhuman actors using the framework, alternative actions or failure cases, and variations on the

use case. The following sections introduce these topics in more detail. (For a complete treatment of use cases, refer to Cockburn [2000], the use cases Web site, or the OOASIS Web site. There are many opinions on use cases and what information they should include and when they should be used. Cockburn is considered one of the top industry sources on use cases. The guidance in OOASIS is based on his work but is somewhat more restrictive and focused).

- **Define framework architecture.** The Define Framework Architecture activity analyzes the software use cases to create static and dynamic views of the framework architecture. The system use cases are further elaborated with details about the overall framework architecture and details of the services. The static and dynamic architectural views are elaborated further with failure clauses, alternative (or recovery) actions, and more detailed interactions. This elaboration and refinement process continues until the framework architecture and elaborated use cases are fully developed.

  The OOASIS methodology provides detailed steps to analyze software use cases and produce the initial framework architecture. Subsequent activities in CAFÉ will use OOASIS methods to further elaborate this architecture based on a generalization of classes and an analysis of deployment characteristics for the framework. Refer to the OOASIS Web site for a more detailed description of these specific activities.

- **Define distribution characteristics.** The static and dynamic views of the architecture represent a logical view of the framework. These views focus on the composition of the framework and relationship between the architecture elements. The physical view of the architecture, also called the deployment diagram, depicts the distribution of classes across physical computing devices. In typical systems development, many issues including system performance, throughput requirements, scalability, and redundancy are considered when allocating classes to physical computing devices. In framework development, the framework architect can anticipate potential distribution schemes based on an expert understanding of the target domain and current trends. There is no means to anticipate all distribution needs of application developers. It is critical that

the distribution characteristics of the framework (including constraints) be accurately and clearly documented.

- **Elaborate framework architecture.** The static and dynamic nature of the framework has been captured and represented in architectural views. Deployment characteristics of the framework classes have been captured and modeled. These three views provide greater insight into the overall structure of the framework than was available during the creation of the initial architecture. Armed with this new insight, the software use cases can be revisited, elaborated, and used to refine the framework architecture.

  Using the static and dynamic views of the framework architecture, the framework architect and development team revisit each of the use cases and add greater detail to the requirement specifications. The main scenario, alternative actions, and variations are all considered and elaborated. The team must be cautious not to introduce design-level information into the software use cases during this elaboration process.

CAFÉ engineering practices are intended to augment the system development practices used in organizations to construct software-intensive systems. These practices address issues of specific interest to object-oriented framework development, which an organization may address with current development methodologies. Additional practices may be needed to institutionalize and improve the efficiency of framework-based development. CAFÉ engineering practices and the associated inputs and outputs are depicted in Table 4.

Table 4. CAFÉ Engineering Activities

Input to / Requires	Engineering Practice	Produces
Business Use Cases Subject Matter/Technical Experts	Define the Domain of the Framework	Summary Use Cases Information Repository/Training
Summary Use Cases Subject Matter/Technical Experts User representatives Stakeholders	Capture Behavioral Requirements for Common Services	Software Use Cases Prioritized List of Services for Development Cycle
Software Use Cases Framework Standards Enterprise Technical Architecture Framework Architect	Define Initial Framework Architecture	Class Diagrams Sequence Diagrams Framework Hot Spots Framework Private Services
Software Use Cases Subject Matter/Technical Experts	Define Distribution Characteristics	Deployment Diagram Class Diagrams Sequence Diagrams
Software Use Cases Deployment Diagram Class Diagrams Sequence Diagrams Framework Standards Enterprise Technical Architecture Framework Architect	Elaborate Framework Architecture	Software Use Cases Class Diagrams Sequence Diagrams

2.2 How CAFÉ Addresses Framework Development Issues

Process solutions, such as those founded on the Software Engineering Institute's Capability Maturity Models, provide generic solutions to engineering and management problems. To be most effective, these solutions must be carefully adapted and tailored to meet specific needs of organizations. Interestingly

enough, technology solutions have the same characteristic – simply purchasing and installing a configuration management tool does not make an organization effective at managing software configurations. The following guidelines[11] refer to generic solutions to the basic framework development issues described in Section 1.1.

2.2.1 Identifying the Proper Framework Services

Intuitively, an OO framework will only be effective if it supplies the proper services and extensions to meet the needs of application developers. Ideally, the framework must also be somewhat agile so that it can change and evolve to meet known and unknown needs. Problem 1 restated below captures the essence of the service identification issue.

> Problem 1. For effective software reuse within industry, engineers require a means to identify, specify, structure, and develop common domain-specific services within their industry.

CAFÉ management activities include a strong emphasis on team composition and early and active stakeholder (including application developers) involvement. Experts in application solution development lead the framework development team and provide the deep experience based required to identify and structure services for the framework. The team also includes strong technical developers who are masters of the development processes and tools used to build the framework and subsequent applications. Together the domain experts and technical experts judicially select services and structure the framework to the best advantage. Most domain and technical experts know the importance of involving all stakeholders, including senior managers, marketing experts, and users (who are developers in this case). Without support from these key groups, the framework may fail even though it is technically sophisticated, powerful, and ideal for building target applications. These groups, in particular the user group, help assure that the framework is developed with the desired services and structured in a manner that is useful for non-experts.[12]

CAFÉ engineering activities emphasize the development of strong business use cases and the subsequent development of system use cases for the framework services. Business use cases capture the operational

[11] As of this writing, CAFÉ has yet to be adopted by a real organization working on real problems. However, the techniques that CAFÉ is based upon have been adopted and utilized within industry for a number of years.

requirements from the organization's perspective of business goals, such as reduction in time to market and quality improvement. These goals identify the specific business domain and business rules (or processes), and defines a scope for the framework. The framework scope identifies the common applications that typify the applications that will be developed using the framework. Subsequent activities refine the business use cases into framework requirements represented as system-level use cases. This refinement is accomplished through domain analysis where applications are examined to identify commonalities, or through an iterative process building, adding, and refactoring the framework.

2.2.2 Optimal Solutions Require Iterative Approaches

For a fixed and known set of target applications, developing a general framework solution as the basis for application development is a non-trivial task. Software product lines identify the variances and commonalities amongst a closed set of applications. The product line can then be used to synthesize any application within the set by selecting the desired characteristics. A framework on the other hand focuses on a known set of example applications, and seeks to develop a generalized solution that is not constrained to a closed set of applications. Unfortunately, this can often lead to developers attempting to build applications beyond the original scope of the framework. There are legitimate reasons, namely pressure to include new features and operations as a result of new technologies in the industry. For frameworks, the solution space is changing and requires and iterative and cyclic process to ensure the framework evolves to meet current needs of developers. Problem 2 restated below summarizes the issue of creating optimal framework solutions.

> Problem 2. Optimal solutions are elusive and require iterative approaches that capture and leverage the experiences of domain and application development experts.

CAFÉ management practices emphasize an iterative development process where stakeholder goals, business needs, and technology drivers are considered and prioritized into the new versions of the framework. Each iteration of CAFÉ translates new business goals, technology advancements, and system enhancements (i.e., defects in the previous version of the framework) into system use cases. System use cases are prioritized and used by developers to implement the next release of the framework.

---

[12] For example, professional cookbooks tend to terse, often just listing ingredients (without measurements) and a brief

2.2.3 Pace of Technology Innovation

It may be trite, but technology will change and frameworks must change to accommodate change. The proper design of any application or system isolates the volatile elements in order to minimize the impact of change on the application. The rapid pace of technology innovation and change taxes both the process of constructing an OO framework and the architecture of that framework. Being prepared for technology change is at least a matter of monitoring and tracking relevant key technologies and understanding when and how they will impact the business and system use cases of the framework. Only by knowing and anticipating these impacts can framework architects hope to properly isolate the proper elements in the framework.[13]

> Problem 4. To keep a framework current and viable, the development process must address activities for identifying new technologies, prioritizing updates, and inserting new technologies into the framework.

In CAFÉ, relevant technologies are monitored and tracked as part of the management activities. When technologies mature or stakeholders needs require the inclusion of a new or updated technology, these changes are cast as system requirements. As system requirements, they are prioritized and scheduled for development just as are new feature requests.

2.2.4 Framework Training

When I started my graduate studies (and before I knew about frameworks), I embarked on learning Microsoft's Visual Studio development environment. It was not too long before I ran head long into the Microsoft Foundation Classes (MFC) and trouble. To be honest, I read only bits and pieces of the ample documentation and none of the excellent books or tutorials. Everything I tried to do caused pain, took days rather than hours, and caused me more than once to search for a decent Java development environment. I eventually broke down and bought a book (Horton 1998) and started to relearn the MFC framework. Almost immediately, things began to click into place, and what took me days of pain suddenly took minutes. This led me to the following observation and a restating of Problem 5.

---

description of the cooking process. They are unusable by amateur chefs, but are fine for professional chefs.

[13] There are technology impacts that are fundamental and cannot be easily isolated, but that is a matter for architects. Eventually, the impact of technology change will drive the framework into obsolescence.

Thesis Conclusion.

OO frameworks have inherent constraints, philosophies and usage models. If developers adhere to these constraints, understand and follow the philosophy and usage models, then they can use the framework to its fullest advantage. Ignoring these rules and models can make the difficult or impossible to use to achieve the desired goals. Don't swim upstream against the current!

Problem 5. To be efficient and effective, developers need ample and sufficient training and knowledge of the engineering design context of the framework. This design context is key to understanding the underlying assumptions on that were built into the design of the framework.

CAFÉ includes a specific activity to capture information pertinent to the training and indoctrination of application developers. The training program captures the business and design rationale, design model, specific interface protocols (parameters, syntax, and semantics), and any additional background information required to fully document the framework. Defects or deficiencies in the training materials are also collected as requirements and prioritized for future versions of the framework. The framework training material also includes reference implementations of applications built using the framework. These reference implementations demonstrate how to use the framework. For example, reference applications show how to implement applications, extend the framework services, supersede existing services, or add new services.

2.2.5 Domain Analysis

Problem-space analysis is key activity successful development methodologies – if you don't identify the problem correctly there is little chance of producing an effective solution and happy customer. For OO framework development, requirements' engineering –the activity of collecting and analyzing user needs – involves handling of multiple requirements sets. In the simplest case, the framework captures the common services from one domain application as a foundation for future applications. The framework team focuses on one application and reengineering one set of requirements. In more complex cases, multiple applications are analyzed, which not only involves multiple analysis activities, but also requires that the team merge the

results into a single unified requirement set. The analysis of multiple domain applications is captured in Problem 1.

CAFÉ focuses initial engineering activities on the analysis of applications in the problem space, i.e., the target domain for the framework. Two major approaches have been tested as part of the overall requirements specification activity. Initially, CAFÉ included domain analysis activities from the Consortium's approach to constructing software product lines (Product Line Management Engineering). This activity essentially decomposes the capabilities of exemplar applications to identify where the applications have commonalities and where there are variances (and the nature of those variances). CAFÉ is primarily interested in identifying those commonalities, but also concerned with some variances. The variances help identify framework hotspots and quantify extensions for the framework.

During the testing of CAFÉ, it became apparent that refactoring is another viable approach to requirements analysis for OO frameworks. Refactoring (Opdyke 1992) is the process of restructuring existing software in order to re-purpose the software for a different application. Refactoring is a viable approach in cases the target set of applications is not initially known, the applications do not exist, or the domain is highly volatile. Using refactoring, an initial framework is developed and deployed. As stakeholders identify new applications or new capabilities, the framework team refactors and restructures to meet the new requirements.

2.3 Summary

Although CAFÉ is still in its infancy, there are several key aspects that can help organizations construct viable frameworks for application development. Complete testing and validation of CAFÉ requires substantial resources in terms of time, staff, equipment, and target domain. We are not able to fully exercise the CAFÉ approach as part of this thesis. Instead we identified what we consider the most crucial portion, the identification of framework services, for our investigation. This focuses on the CAFÉ activity, "Capture Behavioral Requirements for Common Services." The remainder of this thesis recounts the investigation into identifying common framework services.

3 BEHAVIORAL ANIMATION FRAMEWORK

3.1 Introduction

As we have stated throughout this thesis, our target goal was the design of an object-oriented framework to support the development of behavioral animation applications. We have exercised the CAFÉ approach to framework development to identify the proper services and design the framework. During the design of our behavior animation framework, we uncovered insights into the framework design process and developed a substantial understanding of recent research related to behavioral animation. This section provides an annotated description of the CAFÉ Engineering tasks used to design the framework. Key architecture artifacts include the definition of the framework domain, relevant summary business objectives, actors, and the framework services. The final design artifact is an engineering model specified using the Unified Modeling Language (UML). This model is of sufficient detail to support the implementation of the framework using a standard OO development method. For this thesis, we conclude our experiment with the development of an initial software design. Future activities would include an analysis of potential distributions factors and the subsequent iteration on the design. The real nugget; however, is the experience of designing the framework and the experience developing and refining the CAFÉ approach. These nuggets are captured as this thesis, and we have highlighted the brightest of these nuggets.

3.2 Define Framework Domain

In this section, we define the framework domain. That is, we establish the purpose, scope and intended context for the framework. As mentioned above, the domain is important for applications developers to understand the intended context for the framework. Without this context, application developers have little guidance on the proper use and intent of the framework. Figure 5 shows the Define Framework Domain task within the context of the CAFÉ Engineering Tasks.

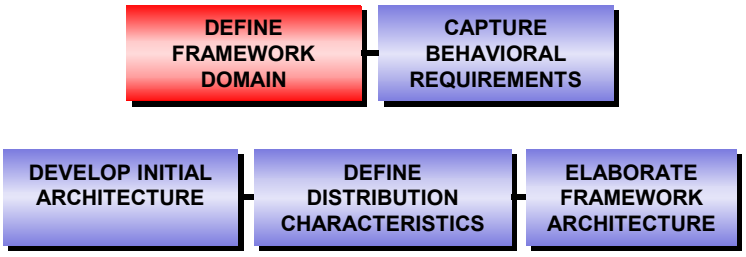

Figure 5 Define Framework Domain Task

3.2.1 Purpose

Behavioral animation applications are a multi-disciplinary approach to developing computer animations using autonomous and semi-autonomous creatures and a varying degree of interaction and control from animators. These animations draw on the knowledge and experiences of researchers in areas such as robotics, artificial intelligence, ethology, and zoology to develop realistic and believable animations and interactive experiences. Given the complexity of these fields, it is common for most behavioral animations to emphasize one or perhaps a few of these research programs.

Computer graphic pioneers Jim Blinn and James Foley have identified the lack of integrated techniques in computer graphics research as one of top ten challenges for the future of computer graphics [Blinn 1999, Foley 2000]. This challenge represents several related issues:

- Lack of a mechanism for sharing techniques without forfeiting the intellectual property of the researcher or researchers
- Difficulty is staying aware of the latest developments from an international community
- Unnecessary duplication of effort that comes from developing similar capabilities

The Behavioral Animation Framework provides common and unique services to support animation development throughout the research community. Common services are identified and developed as a result of an analysis of current research programs in behavioral animation and other closely related programs. These services enable researchers and students to reuse common constructs and focus their design and implementation energy on their particular areas of interest. Unique services capture the technical and domain expertise of researchers and enable other researchers and students to include these unique services in their animation systems. Reusing unique services allows researchers to include addition

techniques in their animations and thereby improving the overall quality and believability of the final product.

A framework development organization is responsible for tracking, capturing and providing these services through an object-oriented framework or set of frameworks. This organization provides central sponsorship, design, implementation, and testing services, framework and application development training, framework compliance and verification rules, and other management functions. Ideally, academic and research institutions would contribute new services to the framework; and in turn utilize the services in the framework.

Several approaches could be used to derive to specify, structure, and populate a framework. Four approaches that have been used to structure available frameworks include:

- ***Program requirements approach*** seeks to identify required functionality in currently deployed systems. The functionality identified is used to structure the framework, and to help specify the population of components. One advantage of this approach is that the framework is more likely to support migration from legacy application to a framework-based application. The major disadvantage appears to be that the framework is unduly constrained by addressing only current technical needs and is limited in its accommodation of future technologies.

- ***Design by committee approach*** seeks to specify framework services in a particular domain by eliciting requirements from industry and academic organization with domain expertise. Through a series of request for proposals and industry/academia responses, the services of the framework are specified. The burden falls on industry to implement the services and provide the population of components for the framework. The advantage of this approach is the use of industry experts and a consensus of appropriate structure and functionality for the framework. The disadvantage is the lengthy process sometimes requiring more than year to develop a specification. This approach, although based heavily on industry involvement, suffers from a lack of commitment by industry to implement products using some specifications.

- ***Standards-based approach*** seeks to leverage existing or developing industry standards or de facto standards to identify required functionality within a domain. While very similar in nature to the

design-by-committee approach, it differs in its leverage of current industry standards. In cases where no accepted industry standards are recognized (or exist), this approach is essentially a design by committee approach. The advantage of this approach beyond the timesavings using existing standards is the focus on developing a reference implementation. The primary disadvantage is the slow speed in which standards bodies develop both standards and reference implementations. In addition, successful standards often have their future closely tied to the success of the associated products from commercial vendors who support the development process. For example, the Object Management Group and the OpenGIS Consortium are successful in developing standards only if their supporting commercial vendors develop products with those standards.

- ***Domain analysis techniques*** are most commonly used as part of either a product line approach to designing software or in the design of reusable library of components. Regardless, domain analysis approaches can serve to help structure and populate frameworks. The essential end product of domain analysis is a model describing common elements within the domain. These elements become the common categories of components within the framework. The main advantage is the development of frameworks in domains without application standards or for corporate product lines. CAFÉ utilizes several of the techniques from the Consortium's Product Line Management Engineering process.

As a result of these analyses, and the use of an iterative development process such as CAFÉ, the behavioral animation framework will continue to evolve through the addition of new services, improvement of existing services, and retirement of obsolete services.

3.2.2 Scope

Behavioral animation applications are a multi-disciplinary approach to developing computer animations using autonomous and semi-autonomous creatures and a varying degree of interaction and control from animators. These animations draw on the knowledge and experiences of researchers in areas such as robotics, artificial intelligence, ethology, and zoology to develop realistic and believable animations and interactive experiences. From the potential broad base of technologies and disciplines to use as an initial

basis, we choose the projects listed in Table 1. The selection of these projects yields a substantial set of capabilities suitable for a particular set of stakeholders or target audience.

There are many potential business and technology stakeholders for our framework and they are represented three industry segments: commercial animation studios, commercial animation technology companies, and education and research institutions.

- **Commercial Animation Studios** develop computer-based or computer-enhanced animations or entertainment/education computer titles. Some organizations are engaged in research and development of computer-generated avatars in conjunction with intelligent agents and collaboration/communication initiatives. Studios might utilize the framework by leveraging the more comprehensive services to develop more compelling and interesting animations and as a basis for to include their own exquisite services.

- **Commercial Animation Technology Companies** develop tools and complete systems used by other organizations to develop animation films and short features. The behavioral animation framework would provide a broader, more comprehensive core element for the development of their unique services.

- **Research and Learning** organizations are academic institutions that provide instruction and research opportunities in areas such as computer animation programming, development of animation titles, and advancing the technical and artistic aspects of computer animation. In short, these institutions are home to the computer science students, art students, and researchers. The framework enables students, faculty and researchers to focus on their unique contributions and still leverage state-of-the-industry contributions.

It is most realistic to target the Research and Learning institutions. The intent of the framework is to consolidate behavioral animation techniques, which would enable students and researchers to develop more interesting animations. In additional, the framework may foster additional research to extend to enhance services.

3.2.3 Context

We envision the Behavioral Animation Framework provide a core, generic application that left unaltered would provide the developer with an executable animation application. We define hotspots for the framework enabling developers to extend and enhance the target animation application. The framework shall be compatible with commercial software development tools, and someday in the future may include a semi-automated "wizard"-like assistant[14].

Successive iterations of the framework will provide more comprehensive capabilities and increased hotspots to enable further customization of the target application. As the framework matures, the core features become the black-box portion of the framework and hide the implementation details of those features. New features are implemented as white-box elements and provide developers with code-level access.

3.3 Capture Behavioral Requirements For Common Services

Once the framework domain has been established, the next step is to identify and capture the behavioral requirements of the framework. These requirements form the services provided to application developers by the framework. The specification of behavioral requirements involves expressing the desired capabilities of the framework from both a system and a software perspective. The goal of these tasks are to determine what services the framework will provide, how users will use those services, and how the services interact with one another. Figure 6 shows how the Capture Behavioral Requirements tasks fits into the Café Engineering tasks.

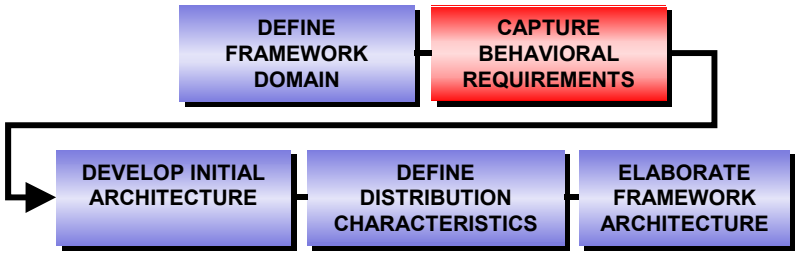

Figure 6 Capture Behavioral Requirements for Common Services

[14] Like the wizards provided in Microsoft's Visual Studio environment and Office applications.

3.3.1 Express Desired System Capabilities as Summary Use Cases

3.3.1.1 Task Objectives

- Describe the workflow processes supported by applications built using the framework.
- Identify which elements are common and will make up the core of the framework.
- Focus the summary use cases on what business function users will accomplish with the applications rather than what user goals they will satisfy.
- Maintain a consistent level of abstraction for the summary use cases by avoiding quantifying or qualifying the processes.
- Answer the business question, "What will the framework help our organization accomplish?"

3.3.1.2 Overview of the Summary Use Cases

An application developed using the framework will consist of several animator commands that control the operation of an application. These commands are used to create, modify and store animation sequences. An animation sequence consists of one or more autonomous creatures interacting among themselves and with their environment. The environment contains dynamic, simple elements, which perform simple actions. An example dynamic element might be a tree falling over. The environment also consists of static elements, such as a bridge, that do not move or perform any actions. The following subsections define the business operations of applications developed using the framework.

3.3.1.3 Create Animation[15]

- Create or open an animation sequence
- Modify animation parameters including character behavior and environment models
- Run animation in test or production mode

3.3.1.4 Dynamic Animated Characters

- Define one or more autonomous creatures[16]

- Define or modify the behavior rules and decision models
- Autonomous relationships between characters
- Autonomous relationships with environment
- Define or modify mobility rules
- Define or modify sensor input rules
- Define or modify physical model

3.3.1.5 Rule-based Environment

- Define or modify "non-player" behavior scripts for dynamic objects
- Define or modify physical models for environment

3.3.1.6 Animation Analysis System

- Record or display animation decisions and behavior rules
- Capture input parameters and drive variations of the animation (i.e., using different decision and behavior models)
- Compare results from multiple "variants" of the animations

3.3.2 Identify Set of Systems for Analysis

3.3.2.1 Task Objectives

- Identify key services by examining representative applications within the domain that meet some or all of the summary use cases.
- Base the selection of systems on those systems that support or contribute elements towards the satisfaction of the system capabilities expressed in the summary use cases.

---

[15] Many frameworks integrate with or are supported by a software development environment. In this case, there will be several summary use cases that describe the business objectives of the framework tool and the goals for the animation developer who uses the tooling.

[16] For simplicity sake, we will limit our framework to one species of creature. Additional summary uses might describe a population of numerous autonomous species – perhaps as part of a doctoral dissertation.

The target framework comprises services that represent the common and essential services as indicated by their presence in the analysis set of systems. The primary systems are listed in Table 1 and summarized starting in Section 1.3.3.

3.3.3 Identify Actors within Each System

3.3.3.1 Task Objectives

- Analyze the identified systems and identify the actors for the system.

Actors are traditionally humans who operate the system to achieve (usually) one of the business goals described in the summary use cases. However, we have chosen to identify non-human actors since the creatures created with our framework must exhibit autonomous behavior. We believe this qualifies them, and the environment as actors. Actors are shown in

Figure 7.

- **Animator/User**. The Animator is the primary human end-user of the system. His or her goals are to develop believable and realistic animations using autonomous creatures.
- **Autonomous creature**. The animated creature or creatures that "live" in the environment as part of the animation.
- **Dynamic Environment**. The environment can support different types of dynamic objects that do not exhibit autonomous behavior but do have action and movement as part of an animation. For example, a geyser that erupts at periodic intervals would be a dynamic object. A "canned" script would be developed and periodically repeated.
- **Static Environment**. The environment also has an obvious static nature that must be modeled and must respond to sensor requests.

The dynamic and static environments are variations of the same actor with the static environment always present and without a dynamic script to enact. We start by modeling them as separate actors, but do not be surprised if that changed during the analysis and design.

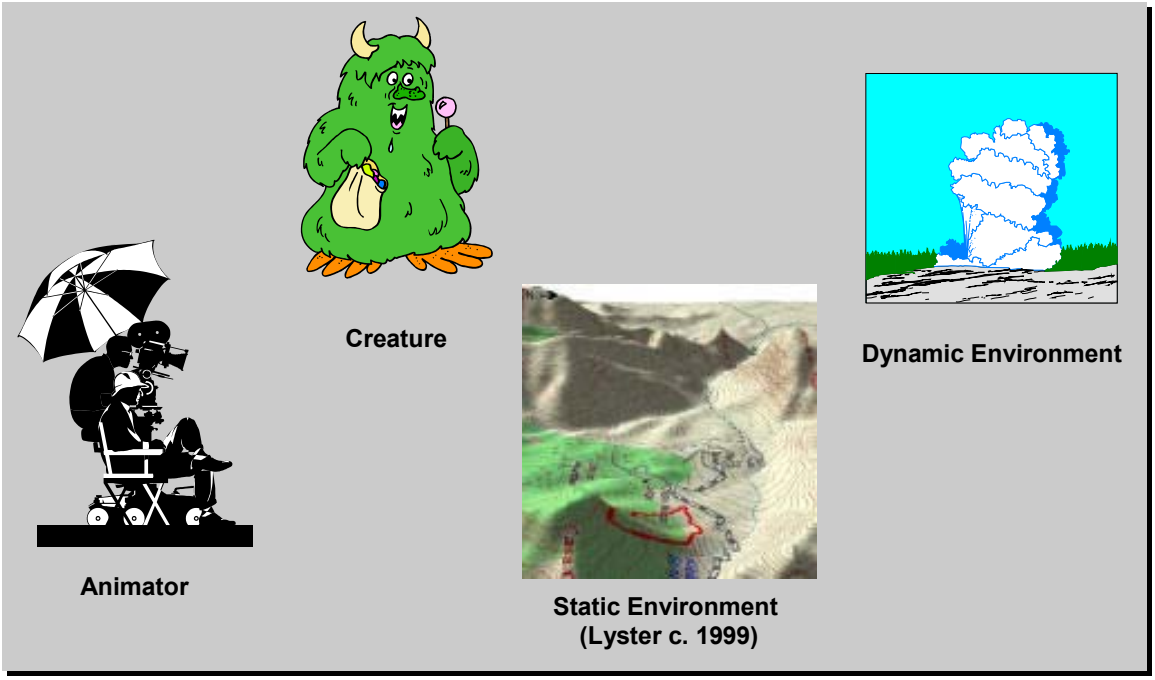

Figure 7. Initial Cadre of Actors

3.3.4 Initial Analysis of Systems for Each Actor

The objective of the Initial Analysis is to identify a master list of the types of methods employed by the actors in each of the systems. The list is not really usefully for designing and implementing a framework, rather it creates a catalog of all the types of methods used in the systems, and the different ways the methods will be used by the actors. From this catalog, the list of common services and potential hotspots will be identified.

Thesis Conclusion.

Each system is decomposed to identify important elements and operations performed by each actor. During the analysis, it is important to remain consistent with level of detail during the analysis. It is often helpful to try and mentally map elements and operations against a standard hierarchy, such as global, national, state, and city or local. During the analysis, identify actions and operations and separate them from attributes and simple attribute setting operations. Build a complete action-oriented model for each system. Try to be as specific as possible when referring to actors and their operations. Do not worry about consolidation, consistent terminology, or consistent naming between systems at this point. Note any special circumstances or details that might be useful in later refinement steps. Also, note references used in the documentation or models of the actor and specific actions and any specialized circumstances.

The initial decomposition of operations and elements are shown in the following tables and figures. Table 5 shows a decomposition of the systems yielding two major classes of operations. *Animation Control* deals with the mechanics of starting and stopping, creating, loading, and storing animation sequences. *Animation Quality and Editors* refers to the manipulation of qualifiers, rules, and other factors that affect the quality and content of an animation sequence.

Table 5. Initial Identification for the Animator Actor

Actor: *Animator*	
Animation Control Sequence Management Sequence Tracing and Debugging	Animation Quality and Editors Behavior Rules Noise Filter Dynamic Environment Rule Creature Attributes Sequence Attributes

Figure 8 and Figure 9 show the decomposition for the Creature actor. The figures show the decomposition of each of the key systems relative to the creature actor. The decomposition shows roughly seven categories of operations for the creatures. *Architecture* generally includes operations and attributes related to the internal structure, operation, and representation of the creature or creatures. *Creature* generally refers to the operations and elements that simulate the life processes of a creature. *Command and Control* refers the interaction between the animator and creature where the animator requests specific operations from the

creature. Theoretically, this could also include creature-to-creature interactions. *Virtual Objects* generally refers to the operations of non-creature objects. *Neural Network* generally refers to the use of a particular construction techniques for simulation thought processes and behaviors. *Behavioral Model* generally refers to various techniques and approaches to modeling the behaviors, objectives, and constraints for creatures. *Emotion Model* generally refers to emotional qualities that alter the thought and behavior process. Note the overlap between many of the categories, for example notice the overlap between the architecture and creature categories.

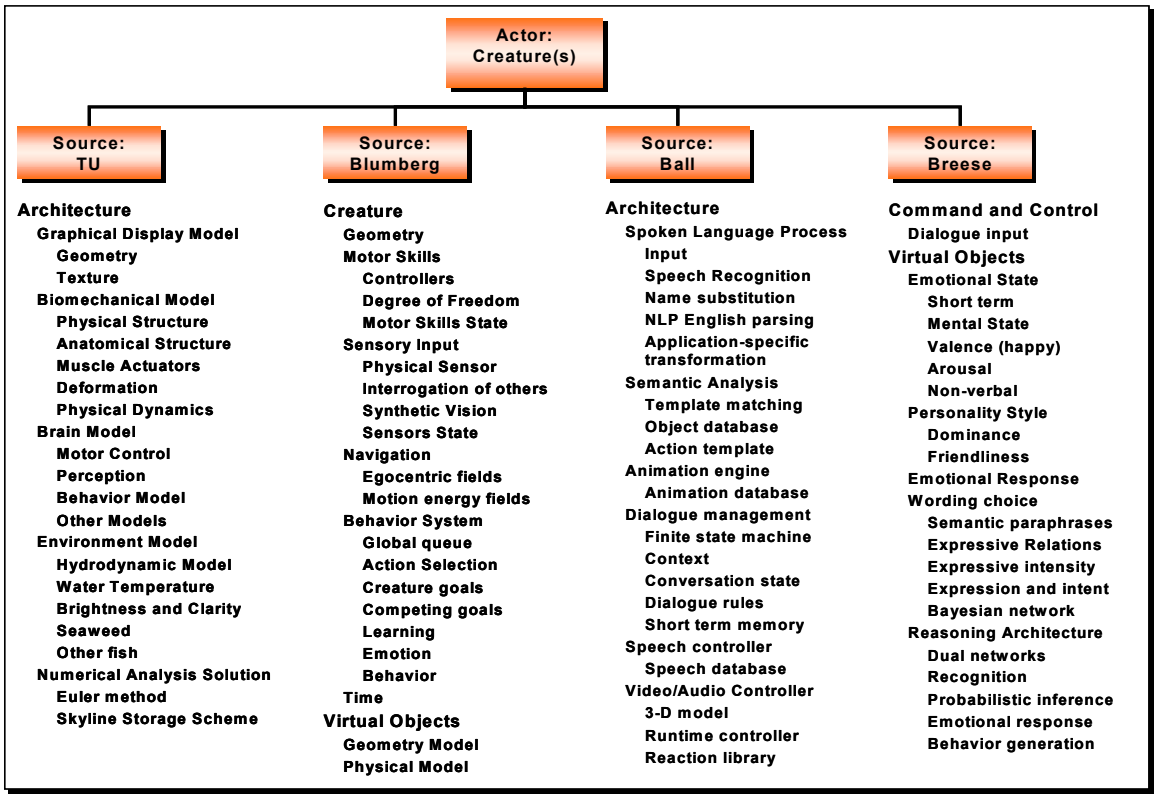

Figure 8. System Contributions for the Actor Creature, Part 1

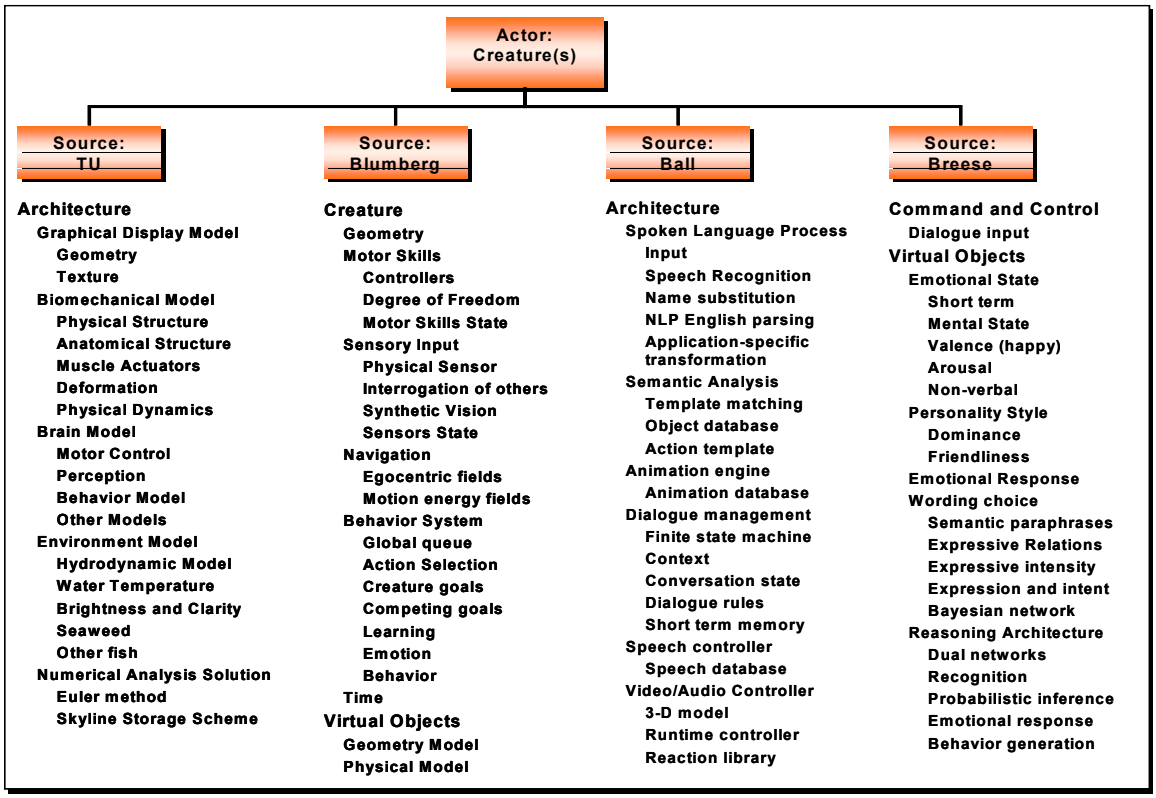

Figure 9. System Contributions for the Actor Creature, Part 2

Table 6 shows the decomposition of the systems for the Dynamic Environment actor. It shows that there are two categories of operations for this actor. The *Physical Model* models and simulates the physical environment or world of the creatures. The *Behavior Model* simulates the behavior and dynamics of the dynamic environment, but is limited to predetermined action patterns (i.e., non-autonomous behavior).

Table 6. System Contributions for Dynamic Environment Actor

Actor: Dynamic Environment	
Physical Model Color Lighting Texture Intersection Geometry Internal Structure	Behavior Model Rule Base Motion Mobility

A similar list for the static environment is shown in Table 7. The initial list of operations focuses on the physical characteristics of the environment such as color, lighting and other elements. The static environment characteristics are drawn primarily from the Tu's work on Artificial Animals [Tu 1996], which describes a physical environment for an aquatic environment. It does include important characteristic

such as collisions between the environment (e.g., rocks) and actors (e.g., fishes), and lighting models which drive sensor perception. Additional resources, such as Virtual Reality Modeling Language and the Geographical Modeling Language could be included as well.

Table 7 System Contributions for the Static Environment Actor

Actor: Static Environment
Physical Model Color Lighting Texture Intersection Geometry Internal Structure

3.3.5 Consolidate Similar Actor Characteristics: First Iteration

As noted above, the operations and attributes identified for the actors across all the systems yielded considerable overlap. The next step is to reorganize these operations and attributes, and in some cases combine specialized actors into more generalized actors. The objective is to construct a refined list of actors and required operations and attributes. This list will drive the development of the system use cases that specify the system-level requirements for the framework.

Thesis Conclusion.

Within each actor, group together similar actor operations or actions to produce a unified set of actions performed by each actor type. Begin by categorizing the major actions or functions of each actor. If possible, combine obvious actors, but be careful that the actors can be generalized together. We found that sometimes the descriptive words used to enumerate the actors' actions in the initial list can be misleading. Consult the additional references and notes you made in the previous step.

Examining the list of operations for the Animator actor (see Table 5), there are no obvious changes operations or generalizations of the actor.

The Creature actor operations can be simplified into 32 categories listed in Table 8. Although this list is a decent consolidation, there are several conflicts. For example, *Behavior Selection* is probably part of the

Behavior System and *Artificial Chemistry* is probably pretty close to *Biochemistry*. The remaining conflicts will be resolving during the next iteration of consolidating the operations.

Table 8 The Initial Consolidation of the Creature Operations

Actor: Creature		
Action Selection	Dialog or speech recognition	Personality
Animation engine	Dialogue management	Reasoning architecture
Artificial chemistry	Emotional response	Scripted Agents
Behavior system	Emotional state	Semantic Analysis
Behavior Selection	Genetics	Sensory input
Biochemistry	Geometry	Sensory Motor Coordination
Biomechanical Model	Graphical Display Model	Speech Controller
Brain Model	Learning system	Spoken Language Processing
Cognitive Model	Motor skills	Structure
Computational Model for behaviors	Navigation	Video/Audio controller
	Numerical Analysis Solutions	Wording selection

The Static and Dynamic Environment actors can be generalized into a single Environment actor. The operations and attributes are similar between the actors with the significant difference being the mobility of the dynamic actor. By setting the mobility of the static elements to zero, the same generalized model can be used for both actors. Table 9 shows the consolidated operations for the Environment actor. The list of operations for the generalized model also contains some conflicts, such as *Behavior Rules and Scripted Behaviors.* These will also be resolved during the next iteration of the consolidation step.

Table 9 The Consolidation of Environment Actors and Operations

Actor: Environment	
Behavior rules	Lighting
Color	Motion / mobility model
Environment Model	Physical Models
Food	Scripted behaviors
Geometry	Time
Geometry	Toys
Intersection model	

3.3.6 Consolidation of Similar Actor Characteristics: Second Iteration

Several of the overlaps and inconsistencies can be resolved by iterating on the consolidation activity. The goal of the consolidation task remains the same, to produce a list of actors and targeted operations as input to the use case development activity. During this iteration, only the Creature and Environment actors require additional analysis.

Thesis Conclusion.

We found during that during the initial consolidation that closely related actors or actions could be identified and consolidated. After the initial iteration, we took a second look at the resultant set and began to question each actor and action. Our goal was to de-conflict the actor and action sets. For example, some researchers emphasize a cognitive model while others allocate the same type of functions to a brain model. Do "cognitive model" and "brain model" represent the same set of functionality? To determine whether or not they do, additional iterations and analysis are required.

Within each actor, group together similar actor operations or actions to produce a unified set of actions performed by each actor type. Begin by categorizing the major actions or functions of each actor. If possible, combine obvious actors, but be careful that the actors really can be generalized together. We found that sometimes the descriptive words used to enumerate the actors' actions in the initial list can be misleading. Consult the additional references and notes you made in the previous step.

There are several elements in operations list that constitute a Brain or Cognitive Model. We combine the operations that simulate the biological, biochemical and genetics mechanisms of brain, the behavioral, and learning models, and the control features such as motor and sensory control. The resulting list is shown in Table 10. We combined S*ensory Motor Coordination, Behavior Selection, Artificial chemistry, Learning system, Structure, Brain Model, Biochemistry, and Genetics* into a *Brain/Cognitive Model*. The resulting consolidation results in a cleaner organization and the identification of six major categories. The *Graphical Display Model* includes the rendering and display of graphical images and presentation of audio streams. The *Biomechanical Model* combines the creature's motor skills, physical geometry model, sensory input,

and navigation models. The *Brain Model* analyzes sensory inputs, applies decision rules according to the cognitive model, and generates the appropriate motor commands. The *Animation Engine* contains the numerical analysis operations and the conversion from motor commands into graphical commands. *Dialogue Management* handles the verbal interactions between creatures and animators and between creatures. It includes the interpretation of utterances and generation of appropriate speech. The *Emotions* operations manage the emotional state and reactions of the creature to particular circumstances and situations.

Table 10 The Second Consolidation the Creature Operations

Actor: Creature		
Graphical Display Model Video/Audio controller Biomechanical Model Motor skills Geometry Sensory input Navigation	Brain Model Brain / Cognitive Model Action Selection Reasoning architecture Behavior system Scripted Agents Animation engine Numerical Analysis Solutions	Dialogue management Speech Controller Dialog or speech recognition Wording selection Spoken Language Processing Semantic Analysis Emotions Emotional state Personality Emotional response

Of these six categories, the *Graphical Display Model* and *Animation Engine* are more generalized and can be associated with other actors. We will remove them and create actors for these capabilities as shown in Table 11.

Table 11 The New Graphical Display Model and Animation Engine Actors

Actor: Graphical Display Model
Video/Audio controller
Actor: Animation Engine
Numerical Analysis Solutions

The Environment Actor comprises both the dynamic and static aspects of the creature's "world". After the second iteration, the actor has been consolidated into three categories. The *Scripted Behaviors* operations manage the dynamics and scripted behaviors for non-creature elements. The *Environment Model* manages the effects of time, weather, and other natural phenomena. The *Physical Model* manages the simulation of

natural characteristics such as lighting models, interactions between creatures and the Environment, and general motion models for non-creature element. The consolidated list of operations is shown in Table 12.

Table 12 Second Consolidation of the Environment Actor.

Actor: Environment	
Scripted behaviors	Physical Models
Behavior rules	Color
Environment Model	Lighting
Time	Intersection model
Environmental conditions	Motion / Mobility Model
Weather (fog, wind, rain, cold, heat)	Geometry
Day light, nighttime	

3.3.7 Develop System Use Cases for Actors and Their Actions

At this point in the CAFÉ approach, the business rationale and objectives for the framework have been described, systems have been surveyed and the resulting actors and their actions have been documented. The summary use cases describe the objectives of the actors and their actions and relate back to the business use cases for the framework. This linkage is useful to ensure that there is a business rationale for the actor, and that the framework meets all business objectives. The next step is to define the semantics of the operations for the actors using system use cases.

The development of use cases is partly analysis and partly black art. Many approaches to describing use cases can be found in various papers and books on object-oriented development. CAFÉ adopts many of the concepts and notions prescribed by Alistair Cockburn (Cockburn 2001). Figure 10 shows the basic relationship between the summary and system use cases. System use cases capture the purpose or goal of the use case that the actor is trying to accomplish by executing the use case. The use cases also list several related scenarios or low-level operations. We also capture failure conditions for the use case and prescribe how these faults should be handled. Use cases also capture variations or alternative scenarios for the use case. A use case variation is a technique that enables framework architects to describe how and where the framework could be extended.

We have documented fifteen system use cases for the system actors in Appendix A. Of these fifteen, the four use cases for the Creature Actor decompose into nineteen additional system use cases. We have also

included in Appendix A simplistic system interface diagrams to show the potential relationship between the use cases for all the actors. This style of diagram depicts the inter-dependencies between use cases and shows how interface potentially flows through the framework. In the design phase for the framework, engineers would use these system use case descriptions to derive the class hierarchy for the framework. The system inter-dependency diagrams help to identify the necessary interfaces between components and help define the hot spots and what variances the framework must accommodate.

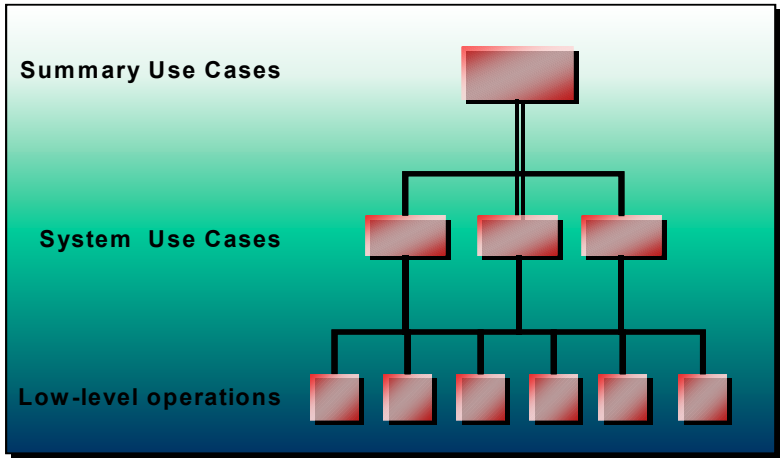

Figure 10 Relationships Between Types of Use Cases

Thesis Conclusion.

System use cases provide the detailed information required to create the architecture.

Define the system use cases for each actor and action. Identify the purpose of the use case, i.e., the goal that the actor obtains by performing the use case. Identify different scenarios for the use case to provide a context and to express any desired constraints. Fault cases express the desired responses to error conditions encountered during normal execution of the use case. List variations to the use case by describing the other possible actions that might augment or replace this use case in the future. The variations help to capture how the framework might be extended through alternative or emerging algorithms. Variations help identify and describe framework "hot spots" that indicate how the framework might be extended or tailored in the development environment.

3.4 Reconcile Summary and System Use Cases

Once the system use cases have been developed to sufficient detail, an analysis can be performed to determine if the system cases support the summary use cases. Table 13 lists the Summary Use Cases from Section 3.3.1 in the rows and the system use cases from Section 3.3.7. The resulting matrix is a coverage matrix that indicates which system use cases correspond to summary use cases.

Thesis Conclusion.

The objective is to ensure that all summary use cases are implemented in system use cases, and that all system use cases support at least one of the summary use cases. When reviewing the coverage matrix, an empty row indicates that no system use case implements that summary use case and that the system will not meet that requirement. An empty column indicates that a system use case exists for which there is no corresponding summary use case (i.e., there is no requirement).

During the analysis of the coverage matrix, we discovered that the Creature Actor contains a use case for Dialogue Management. In the initial set of summary use cases, there was no requirement for creatures to communicate between themselves. To simplify the framework, the Dialogue Use Cases could have been dropped since no requirement existed. We chose to make the framework more comprehensive and revisited the summary use cases.

Table 13. Reconciliation Between System and Summary Use Cases

System Use Cases / **Summary Use Cases**	Animator			Character				Environment			Graphical Display			Animation Engine	
	Animation Sequence	Animation Editor	Execution Mode	Biomechanical Model	Brain Model	Dialogue Mgmt	Emotions	Timers	Dynamics	Static	Rendering	Debugger	Sequence	Math Support	Management
Create Animation															
Create	■	■													■
Modify	■	■													■
Production	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Debug			■								■	■	■	■	
Dynamic Character															
Multiple Creatures	■	■					■	■					■		■
Modify behavior rules	■	■			■		■	■							■
Modify Mobility	■	■		■						■					■
Modify Sensors	■	■			■					■					■
Modify Physical Model	■	■		■					■	■					■
Modify Inter-creature Interactions						■	■						■		■
Rule-based Environment															
Define Non-creature scripts	■	■						■							■
Modify physical environment	■	■							■						■
Animation Analysis															
Record animation decisions and behaviors	■		■									■	■		
Capture/modify input parameters		■	■												■
Compare results from multiple animations			■									■			

Once the Summary and System Use Cases have been reconciled, the initial system architecture can be developed. There are numerous approaches for constructing the initial architecture from use cases. The CAFÉ approach is independent of the specific design approach. The approach taken here is to develop for each primary actor an aggregate class comprised of classes that provide the functionality listed in the use

cases. The variations listed for the use cases are modeled into the classes and become hot spots for the framework.

3.5 Develop Initial Architecture

The goal of the Develop Initial Architecture activity is to create the initial system structure and relationship as a first step towards the software design and analysis activities. The Develop Initial Architecture is an iterative activity that may require several iterations before an acceptable structure is obtained. Figure 11 shows the Develop Initial Architecture as part of the CAFÉ approach.

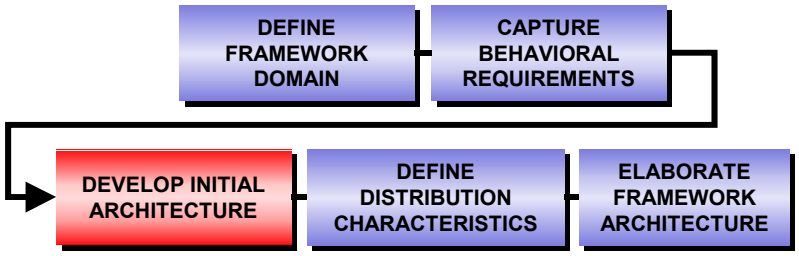

Figure 11. CAFÉ's Develop Initial Architecture Activity

To create the initial architecture shown in Figure 12, create a class for each of the actors. An animation sequence will be essentially a system that passes messages (which equate to object invocations) between objects. For example, when the animator starts the execution of a sequence, the CAnimator class sends an initialization message to the CAnimationEngine class. Create each actor is a composition of supporting classes along with its own class specific operations. These secondary classes are unique to their associated actor classes, so an alternative modeling approach is also appropriate. For example, Figure 13 shows an alternative model for the Animator Actor. The major difference is whether to separate out the execution and sequence models into secondary classes. The typical reason for such a separation is if there are multiple classes that can reuse the same secondary class. Other reasons are to manage the size and complexity of actor class. Our model separates secondary classes for manageability.

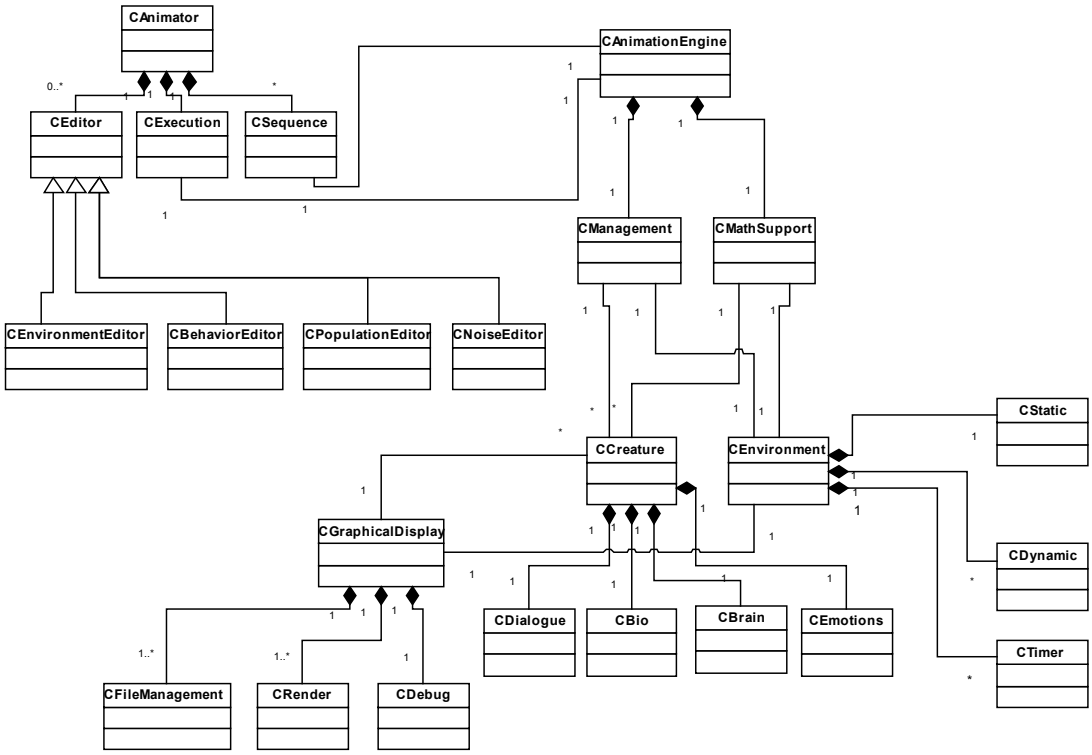

Figure 12 Initial Architecture Class Diagram

The Animator actor class also uses inheritance for the multiple editor functions. A core set of editor functions are defined in the general editor class, and specialized in the secondary editor classes. A similar structure would be used for multiple creatures with similar physiologies.

Thesis Conclusion.

Create a secondary class for each set of use cases for the actor class. Indicate the cardinality of the relationship between actor and secondary class, that is, show whether there is a one-to-one, one-to-zero, or one-to-many relationship. Add associations to indicate the relationships between classes within the system. It is acceptable for the list of associations between classes to be incomplete at this point in the process. Future iterations and refinements will continue to identify new associations and remove any false associations.

Hot spots are the points of variation in the framework. Any class with public visibility is the source of variation for the framework. Application developers can subclass, inherit, or overload any of the public classes. For example, the intent of this version of the framework was to create one creature "specimen" and

not allow any variations. The Creature Actor class should be a private class. If however, we wished to allow for multiple species of creatures, the Creature Actor would be modeled as a public class which would allow developers to overload or extend the Creature Actor class.

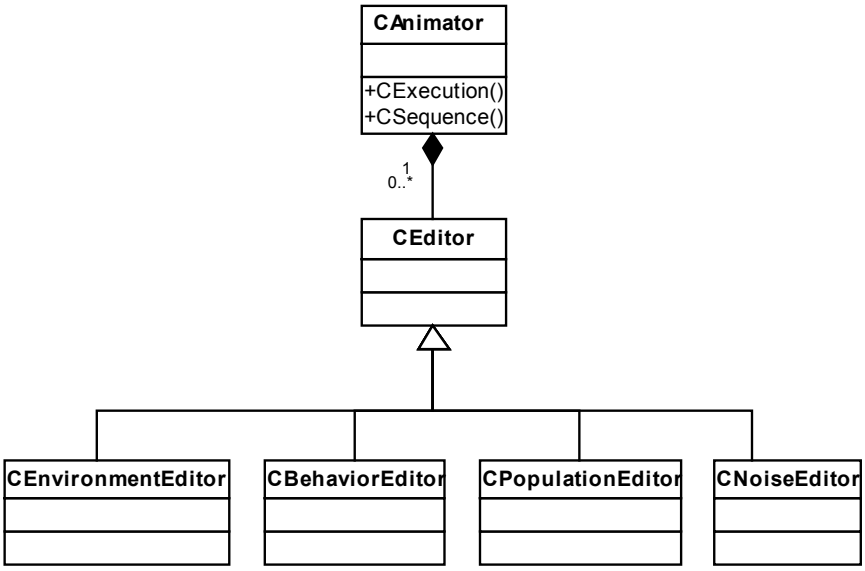

Figure 13 An Alternative Model of the Animator Actor

The architecture shown in Figure 12 describes a white-box, object-oriented framework. With no extensions or overloading of classes, developers would use the framework to create an application that provides a basic behavior animation application. The capabilities of this default system will be consistent with the system use cases and the summary use cases. If we have done our analysis correctly, this default system is representative basis for animation applications in the target domain and representative of the research programs that contributed to it.

The framework has the potential for a number of hot spots that correspond to the variations described in the use cases. Developers can exploit these hot spots by extending current functionality through inheritance or replacing current functionality by overloading existing classes. Since CAFÉ is an iterative and increment development approach, we can be judicious and enable only a few hot spots at a time.

4 FUTURE EFFORTS

This section reflects on the efforts to date to design an animation framework. It describes some potential future areas of development for this thesis effort, and describes potential applications in other domains. There are a number of viable domain areas where the development of objective oriented frameworks could help organizations produce applications based on a standardized set of core services.

4.1 Reflections on Accomplishments

Accomplishments relative to this thesis relate to the application of the CAFÉ approach to the development of object-oriented frameworks and the analysis of behavioral animation research to yield a target domain. The CAFÉ approach was based on the concept of adopting in piecemeal fashion elements of existing methods for common software engineering tasks. One of the collateral results of our work was to validate to some degree the concept of weaving method parts into new approach. We found that the method parts flowed nicely leveraging and supported by OO design concepts. Upon further reflection, the initial steps of CAFÉ are independent of the ultimate conclusion of developing and OO framework. These steps are equally valid for the analysis of legacy and heritage systems as part of a larger reengineering effort.

The consolidation of several similar but disparate systems into a single analysis model presents several challenges. The foremost of these challenges is the semantic interpretation of the systems themselves. One of the side effects of using research programs is the predisposition of documentation (i.e., the dissertations) to be highly explanatory[17]. Still, different research use different terminology for the same or very similar concepts. Tracing between these similarities while not glossing over distinctions is a complicated exercise. The iterative nature of CAFÉ enables developers to reanalyze systems to ensure the more accurate interpretation of the information. We saw this in Section 3.3.6 when trying to consolidate similar actor characteristics.

It is difficult avoid typical software engineering problems related to requirements gathering, notably requirements creep. We fell into this seductive trap by adding the alluring Emotion Actor to the framework. The inclusion of an emotional model was not a service initially targeted for the framework. During the

analysis of the cognitive model, I read about the concept of emotions altering the decision process. The field of emotional modeling is quite compelling and managed to slip into the framework design, albeit in a rather elementary fashion. Unfortunately, there is no silver bullet in CAFÉ that will slay the problem of requirements creep.

It is intuitive, and consistent with the SPIR principle[18] that domain expertise with the target systems is critical to the successful analysis of the systems. Domain expertise enables the far deeper comprehension of the systems including its nuances than is possible from documentation. However, one of the more surprising and frustrating discoveries was the criticality of technical expertise with development tools. For the most part, I have been using Visual Studio as a simple C language compiler and debugger and had never really delved into the proper use of the tool to build Windows-based applications. Mastering the development environment ultimately required halting the framework analysis work for a substantial period of time, but this time was invaluable in understanding the context in which the framework would be used by developers. This context is the basis for understanding the implementation (and hence design considerations) for framework hot spots.

Despite the fact that the research programs we analyzed emphasized different aspects of autonomous creatures, artificial life, and behavioral modeling, we were able to find a central core of services that appear to be common among these systems. Using CAFÉ, we were able to synthesize other non-behavioral animation systems, such as the Conversational Agent research, into the framework with little additional effort.

During the course of this research, I learned a great deal about many things. I definitely improved my knowledge and understanding of OO concepts, OO design methods (in particular use cases), and OO programming (in particular Microsoft Windows programming). I learned a great deal about behavioral modeling, cognitive modeling, ethology, fish, speech recognition, different models of learning, and emotional modeling. I learned a great deal about developing and testing processes and methods. I learned a

[17] Doctoral candidates try very hard to explain and convince others of the merits of their system. Their writing is far more detailed then most industry documents I have read.
[18] Smart People In a Room (SPIR) is a term I coined at the Consortium. It reflect an observation that most if not all analysis and evaluation techniques, methods, and processes include, sooner or later, a step where the team gathers all the experts into a room. Once gathered, the experts can distill their knowledge to the rest of the team.

great deal about the world of frameworks, OO frameworks in particular, but also about architecture frameworks, application frameworks, and product lines.

4.2 Next Steps

The CAFÉ Approach describes two major activities beyond the Developing the Initial Architecture activity. These are shown in red in Figure 14. The Define Distribution Characteristics is an analysis of the initial framework architecture to determine how to best partition the framework for distribution across multiple computer systems. The intent of this step is to build in the infrastructure mechanisms that would allow, for example, the Static Map and Dynamic Elements of the Environment Model to be transparently distributed across a network. The CAFÉ activity for performing this analysis has not yet been completed, but is likely to revolve around a SPIR principle.

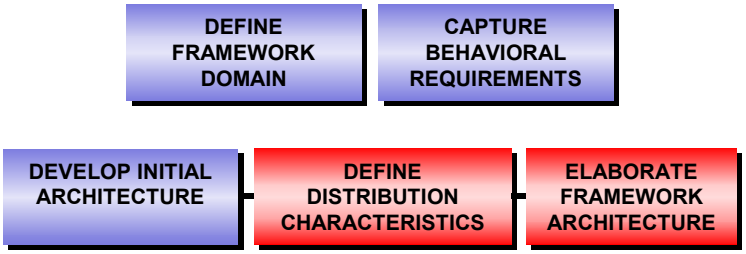

Figure 14 Future CAFÉ Activities

The Elaborate Framework Architecture task is another iteration of analysis similar to developing the initial architecture. For this activity, the initial architecture is considered along with the distribution characteristics to determine the next of level of detail for the framework architecture. The goal is to describe the structure of the framework as a driver for the software design process. This CAFÉ activity has not yet been developed.

Once the elaborated architecture has been developed, and the framework implemented, it must be integrated closely into development environment to simplify development of animation applications. Without this integration, the repetitive use of a framework is difficult with high-end tools such as Visual Studio.

The framework services themselves provide ample opportunity for future work. Each variation listed in the use cases can be used to extend the functionality and capability of the framework. However, during the analysis of systems that drove the framework construction three major areas of interest emerged. These areas are quite complex, and are the focus of numerous research programs. These areas are summarized in Table 14.

Table 14 Future Extensions for the Behavioral Animation Framework

Extension Area	Description
Comprehensive Learning Model	Learning is complex. Many researchers are investigating how creatures learn, different modes of learning and how they are applied in different circumstances, the effects of short and long-term memory, the effects of experiences, and other phenomena. Realistic creatures utilize realistic learning models.
Comprehensive Emotions Model	We found that emotions are a complex and controversial subject. There is a substantial amount of research on the modeling of emotions, their effects on decisions and behaviors, and emotional problems.
Society Model	The effect of society is a critical element of any animation with a substantial population. These effects might include “the gang mentality”, “peer pressure”, “safety in numbers”, and other similar effects.

4.3 Alternative Domains and Framework Uses

In the domain of biomedical simulation, significant research has been conducted at numerous research institutions. Motivations for simulating the processes and functions of the human body are varied and include decreased testing time for new drugs, more complete testing of drugs on varying physical conditions, and a reduction in the amount of animal-based testing. These research programs were attempting to model the human body functions with the goal of producing a simulation environment capable of testing the short and long-term effects of new drugs on humans. My particular interest in this research was their goal in reducing the amount of drug experimentation on animals. The simulation

environment researchers were trying to achieve was far too complex for the software and hardware technology at the time, and maybe even with today's technology. Still, this research is the spark for the following example and potential future project.

Envision a simulation environment where the human body is modeled using hundreds or thousands of autonomous agents. Each agent models a particular element of the body, for example capillaries, veins, organs, or muscle tissue. The agents are programmed with their own particular "business rules" and know their objectives and how to interact with other agents to achieve their goals. Each agent also contains a set of basic services, such as communication services and message interpretation services. I envision building a framework produces a generic "body element" agent, and that this framework contains hot spots to enable the specialization of agents to simulate specialized body elements. Using hot spots, developers can also model various defects and degenerative conditions for body elements. For example, the framework would produce an agent that represents a healthy and normal liver, while hotspots within the framework would allow the user to create diseased, aged, or malformed livers. Each "liver agent" contains the internal services that mimic the flow of signals, fluids and other "liver processes".

Once enough agents have been developed to simulate a body environment, agents representing drugs can be inserted into the stable agent system. The emergent behavior models the effects of the drug on the body. Variations in specific body-element agents can be used to test the drug on various body types under various conditions (e.g., sick, tired, intoxicated, pregnant).

List of References

Andre, Elisabeth, Thomas Rist, and Jochen Muller
1998
Integrating Reactive and Scripted Behaviors in a Life-Like Presentation Agent. Saarbruken, Germany: German Research Center for Artificial Intelligence.

Badler, Norman I., Cary B. Phillips, and Bonnie L. Webber
1992
*Simulating Humans: Computer Graphics, Animation, and Control.*New York, New York:Oxford University Press.

Ball, Gene, Dan Ling, David Kurlander, John Miller, David Pugh, Tim Kelly, Andy Stankosky, David Thiel, Maarten Van Dantzich and Trace Wax
1997
Lifelike Computer Characters: the Persona project at Microsoft Research. Redmond, Washington:Microsoft Corporation. [Online] URL http://research.microsoft.com/research/ui/persona/chapter/chapref.htm

Bargen, Bradley and Peter Donnelly
1998
Inside DirectX. Redmond Washington:Microsoft Press.

Bates, J.
1994
"The Role of Emotion in Believable Agents*." Communications of the ACM*. 37(7), 1994. pp 122-125.

Bayer, Joachim, Dirk Muthig, and Tanya Widen.
1999
"Customizable Domain Analysis." Location, Kaiserslautern, Germany:Fraunhofer Institute for Experimental Software Engineering. [Online] URL http://www.netobjectdays.org/pdf/stja/bayer.pdf

Becheiraz, Pascal and Daniel Thalman
1998
"A Behavioral Animation System for Autonomous Actors personified by Emotions." In *Workshop on Embodied Conversational Characters.* Palo Alto, California: FX Palo Alto Laboratory. pp 57-65.

Blinn, Jim
1999
"SIGGRAPH 1998 Keynote Address." In *ACM SIGGRAPH -Computer Graphics.* New York, New York:Association for Computing Machinery. pp 43-47

Bosch, Jan, Peter Molin, Michael Mattsson, and PerOlof Bengtsson
1999
"Object-oriented Frameworks – Problems & Experiences." In *Object-Oriented Application Frameworks.* New York, New York John Wiley & Sons.

Booch, Grady, James Rumbaugh, and Ivar Jacobson
1999
The Unified Modeling Language User Guide. Reading, Massachusetts: Addison-Wesley. Page 457.

Blumberg, Bruce Mitchell
1997
Old Tricks, New Dogs: Ethology and Interactive Creatures. PhD Thesis. Cambridge Massachusetts: Massachusetts Institute of Technology.

Blumberg, Bruce M. and Tinsley A. Galyean
1995
Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments. Cambridge, Massachusetts:Massachusetts Institute of Technology.

Binsted, Kim
1998
"Designing Portable Characters." In *Workshop on Embodied Conversational Characters.* Palo Alto, California: FX Palo Alto Laboratory. pp 77-87.

Bowman, Ivan
1998
An Architectural Investigation: Extracting Architectural Facts From a Large Software System. Waterloo, Ontario: University of Waterloo. [Online] URL http://plg.uwaterloo.ca/~itbowman/CS746G/proj/Project.html

Breese, Jack and Gene Ball
1998
Modeling Emotional State and Personality for Conversational Agents. MSR-TR-98-41. Redmond, Washington:Microsoft Corporation.

Canamero, Dolores
1997
"Modeling Motivations and Emotions as a Basis for Intelligent Behavior." In *Proceedings of the First International Conference on Autonomous Agents.* Marina del Rey, California: ACM Press. pp.148-155

Cassell, Justine, Yasmine B. Kafia, and Mary Williamson
1997
"The Implications of a Theory of Play for the Design of Computer Toys." In *Computer Graphics Proceedings*. New York, New York:Association for Computing Machinery. Pp 431-433.

Cheng, Lili
1999
Microsoft Virtual Worlds v1.5 Designers' Guide for 3D Worlds. Redmond, Washington:Microsoft Corporation.

Chicago
1982
The Chicago Manual of Style .Chicago, Illinois:University of Chicago Press.

Cockburn, Alistair
2001
Writing Effective Use Cases. Upper Saddle River, New Jersey:Addison-Wesley.

De Champeaux, Dennis, Doug Lea and Penelope Faure
1995
"Domain Analysis. Chapter 13 In *Object-Oriented System Development*. Reading, Massachusetts: Addison-Wesley. [Online] URL http://g.cs.oswego.edu/dl/oosdsw3/ch13.html

Foley, James
2000
"Getting There: The Top Ten Problems Left." *IEEE Computer Graphics and Applications.* 20:1, 2000. pp. 66-68.

Funge, John David
1998
Making Them Behave. Cognitive Models for Computer Animation. PhD Thesis. Toronto, Canada:University of Toronto.

Funge, John, Xiaoyuan Tu, and Demetri Terzopoulos
1999
Cognitive Modeling: Knowledge, Reasoning, and Planning for Intelligent Creatures. In *Proceedings of SIGGRAPH 99.* 1999. New York, New York: Association for Computing Machinery. pp 205-216.

Gordon, Diane F.
1998
"Well-Behaved Borgs, Bolos, and Berserkers." In *Proceedings of the 15th International Conference on Machine Learning*. San Francisco, California: Morgan Kaufman Publishers

Grand, Stephen, Dave Cliff, and Anil Malhotra
1997
"Creatures: Artificial Life Autonomous Software Agents for Home Entertainment." In *Proceedings of the First International Conference on Autonomous Agents.* Marina del Rey, California: ACM Press. pp. 22-29

Gritz, L. and J. K. Hahn
1995
"Genetic Programming for Articulated Figure Motion." *The Journal of Visualization and Computer Animation.* 6:3 pp.129-142.

Goodall, Jane
1971
*In the Shadow of Man.*Houghton Mifflin Company:Boston Massachusetts.

____
1990
Through a Window. Houghton Mifflin Company:Boston Massachusetts.

Hayes-Roth, Barbara, Lee Brownston, and Erik Sincoff
1995
Directed Improvisation by Computer Characters. KSL-95-04. Palo Alto, California:Stanford University.

Hahn, James
1997
*"*Behavior Animation" CS 206 Computer Animation Course Notes. Washington D.C.:George Washington University.

Horton, Ivor
1998
Beginning Visual C++ 6. Birmingham, United Kingdon:Wrox Press Limited.

Kang, Kyo C., Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson.
1990
Feature-Oriented Domain Analysis (FODA) Feasibility Study. CMU/SEI-90-TR-21. ESD-90-TR-21. Pittsburg, Pennsylvannia:Software Engineering Institute.

Kim, Hyseob and Cornelia Boldyreff
1997
"Formalizing Design Patterns and Frameworks: A Survey Report." Technical Report 97/2. Durham, United Kingdom:University of Durham.

Kovach, Peter J.
2000
Inside Direct3D. Redmond Washington:Microsoft Press.

Hodgins, J.K. and W.L. Wooten
1998
"Animating Human Athletes." [Online] URL http://www.cc.gatech.edu/gvu/animation/papers/isrr.ps.gz Atlanta, Georgia:Georgia Institute of Technology.

Isla, Damian, Robert Burke, Marc Downie, and Bruce Blumberg
No date.
"A Layered Brain Architecture for Synthetic Creatures." [Online] URL http://web.media.mit.edu/~solan/layeredArchitecture.pdf
Cambridge, Massachusetts: Massachusetts Institute for Technology.

Lewandowski, Scott M.
1998
"Frameworks for Component-Based Client/Server Computing." *ACM Computing Surveys.301.* pp3-27

Liu, Zicheng and Michael F. Cohen
1995
*An Efficient Symbolic Interface to Constraint Based Animation Systems.*MSR-TR-95-27. Redmond, Washington:Microsoft Corporation.

Lyster, Alec
c. 1999
Lyster, Alec and Greg Murdoch. The Casa Grande Burn - Recognizing Critical Fire Weather Patterns. National Weather Service Midland/Odessa, Texas. [Online] URL http://www.srh.noaa.gov/maf/html/Alecfwxi.htm

Maes, Pattie
1989
How to Do the Right Thing. Cambridge, Massachusetts. Massachusetts Institute of Technology. [Online] URL
http://agents.www.media.mit.edu/groups/agents/publications/Pattie/consci/

____
1994
*"*Modeling Adaptive Autonomous Agents." *Journal of Artificial Life.* 1:1/2. Cambridge, Massachusetts:MIT Press.

____
1995
"Artificial Life Meets Entertainment: Lifelike Autonomous Agents." *Communications of the ACM*. 38:11. Pp. 108-114.

Mataric, Maja J., Victor B. Zordan, and Mathew M. Williamson
1999
"Making Complex Articulated Agents Dance." *Autonomous Agents and Multi-Agent Systems*. 2,1999. pp. 23-43.

Microsoft Corporation
1998
Designing Characters for Microsoft Agent. Redmond, Washington: Microsoft Corporation.

O'Neill, Barry
2000
Approaches to Modeling Emotions in Game Theory. DRAFT, Palo Alto, California:Stanford University. [online] URL http://www.stanford.edu/~boneill/emotions.html

Opdyke, William F.
1992
Refactoring Object-Oriented Frameworks. PhD Thesis. Urbana, Illinois:University of Illinois at Urbana Champaign.

Perlin, Ken and Athomas Goldberg
1996
"Improv: A System for Scripting Interactive Actors in Virtual Worlds." In *Proceedings of SIGGRAPH 96.* 1996. New York, New York: Association for Computing Machinery. pp 205-216.

Perlin, Ken
1995
"Real time responsive animation with personality." *IEEE Transactions on Visualization and Computer Graphics*. 1:1. Pp 5-15.

Pisanich, Greg and Michael Prevost
1996
"Representing Human Characters in Interactive Games." In *Proceeding of the Computer Game Developers Conference.* San Francisco, California:Miller-Freeman, Inc.
[Online] URL http://reality.sgi.com/prevost _studio/personality.htm

____
1997
"Representing Artificial Personalities." Presented at *1997 Computer Games Developers Conference.* Santa Clara, California.
[Online] URL http:// reality.sgi.com/prevost *studio/GDC97*_paper.htm

Pree, Wolfgang
1995
Design Patterns for Object-Oriented Software Development. Wokingham, England: Addison-Wesley.

Reynolds, Craig
1987
"Flocks, Herds, and Schools: A Distributed Behavior Model." In *Computer Graphics Proceedings Annual Conference Series, 1987.* pp. 25-34.

Richardson, Stephen D., William B., Dolan, and Lucy Vanderwende
1998
MindNet: acquiring and structuring semantic information from text. MSR-TR-98-23. Redmond, Washington:Microsoft Corporation.

Roberts, Don and Ralph Johnson
1996
Evolving Frameworks: A Pattern Language for developing Object-oriented Frameworks, Urbana-Champaign,Illinois:University of Illinois. [Online] URL http://st-www.cs.uiuc.edu/~droberts/evolving.pdf

Rogerson, Dale
1997
Inside Com. Redmond, Washington:Microsoft Press.

Sims, Karl
1994
"Evolving Virtual Creatures." In *Computer Graphics Proceedings Annual Conference Series, 1994.* New York, New York: Association for Computing Machinery. pp.15-22.

Schneider, Phillip J. and Jane Wilhelms
1998
Hybrid Anatomically Based Modeling of Animals. USCS-CRL-98-05. Santa Cruz, California:University of California, Santa Cruz.

Sharp, David C.
2000
Containing and Facilitating Change Via Object-oriented Tailoring Techniques. The Boeing Company: St. Louis Missouri.

Schmidt, Hans Albrecht
1997
"Systematic Framework Design by Generalization." Communications of the ACM 40,10:39-42.

Strassman, Steven Henry
1991
Desktop Theater: Automatic Generation of Expressive Animation. PhD Thesis. Cambridge, Massachusetts. Massachusetts Institute of Technology.

Strauss, Paul S.
1988
BAGS: The Brown Animation Generation System. PhD Thesis. Providence, Rhode Island. Brown University.

Szyperski, Clemens
1998
Component Software : Beyond Object-Oriented Programming. Reading, Massachusetts: Addison-Wesley. Page 366.

Thomas, Frank and Ollie Johnson
1981
The Illustration of Life. Disney Animation. New York, New York: Hyperion.

Template University
1997
"Bayesian Networks." In *SUMMER WORKSHOP FOR EDUCATORS "Providing and Integrating Educational Resources for Faculty Teaching Artificial Intelligence"* [Online] URL http://yoda.cis.temple.edu:8080/UGAIWWW/lectures/bnets.html

Terzopoulos, Demetri, Tamer Rabie, and Radek Grzeszczuk
1996
"Perception and Learning in Artificial Animals." In *Artificial Life V: Proceedings Fifth International Conference on the Synthesis and Simulation of Living Systems.*

Tu, Xiaoyuan and Demetri Terzopouls
1994
"Artificial Fishes: Physics, Locomotion, Perception, Behavior." *In SIGGRAPH 94 Conference Proceedings*. New York, New York: Association for Computing Machinery.

Tu, Xiaoyuan.
1996
Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior. Toronto, Canada:University of Toronto.

Unuma, Munetoshi, Ken Anjyo, and Ryozo Takeuchi
1995
Fourier Principles for Emotion-based Human Figure Animation." In *Computer Graphics Proceedings Annual Conference Series, 1995.* New York, New York: Association for Computing Machinery. Pp 91-96.

Vellon, Manny, Kirk Marple, Don Mitchell, and Steven Drucker
1998
"The Architecture of a Distributed Virtual Worlds System.." In *The Fourth Conference on Object-Oriented Technologies and Systems (COOT) Proceedings*. Berkeley, California:USENIX Association. pp 211-218

Wang, Guijun, Liz Ungar, and Dan Klawitter
1999
Component Assembly for OO Distributed Systems. IEEE Computer 32,7:71-78

Wang, Yi-Min, Wilf Russell, Anish Arora, Jun Xu, and Rajesh K. Jagannathan
2000
Towards Dependable Home Networking: An Experience Report. MSR-TR-2000-26. Redmond, Washington: Microsoft Corporation.

Webster's II
1984
Webster's II New Riverside Dictionary. New York, New York:Berkley Publishing Group.

Wermter, Stefan, Jim Austin, David Willshaw, and Mark Elshaw

2001

"Towards Novel Neuroscience-inspired Computing." In *Emergent Neural Computational Architectures based on Neuroscience.* Heidelberg, Germany York:Springer-Verlag

Wilhelms, Jane and Allen Van Gelder

1997

Anatomically Based Modeling. UCSC-CRL-97-10. Santa Cruz, California:University of California, Santa Cruz.

Yang, Young Jong, SoonYong Kim, Gui Ja Choi, Eun Sook Cho, and Soo Dong Kim

1998

"A UML-based Object-Oriented Framework Development Methodology." In *Asia-Pacific Software Engineering Conference*. New York, New York:IEEE Computer Society. pp 211-218

Zambonelli, Franco, Nicholas R. Jennings, Andrea Omicini, and Michael Wooldridge

2000

"Agent-Oriented Software Engineering For Internet Applications." Chapter 13 in *Coodination of Internet Agents: Models, Technologies, and Applications.* Heidelberg, Germany York:Springer-Verlag..

APPENDIX A SYSTEM USE CASES

A.1 Animation Actor Use Cases

The Animator is the principle actor using the framework to create and modify animation sequences. The animator can also modify the parameters of the sequences using a variety of editors. The relationships between the elements of the Animator Actor are shown in Figure 15.

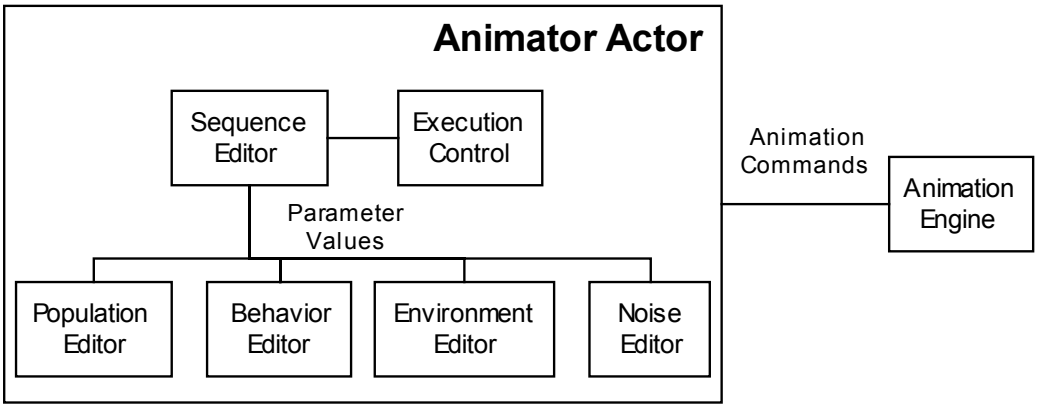

Figure 15 Relationships Between Animator Use Cases

A.1.1 Modify Animation Sequence

Purpose: The animator opens a new sequence. The system stops and closes the current sequence offering to save any changed work. The system clears the user interface and the animation display. The system displays the environment variable and the sequence definition editors. The animator closes the current animation sequence. The systems stops the current sequence, offers to save or save as if the environment has changed, prompts for file location, prompts for description if new file, writes the environment and other pertinent information to indicated file, clears the user interface, clears the animation display, reports status to the animator. The animator reviews the current animation sequence. The system stops the current sequence, rests the environment variables, clears the animation display, and begins to play the sequence.

Scenario: Animator creates a new sequence or opens an existing one. The animator stops and closes the current animation sequence. The animator deletes existing sequences.

Fault Cases: The system experiences file I/O errors opening the sequence file. The sequence file opens successfully but does not contain properly formatted data.

Variations: No variations currently identified.

A.1.2 Manipulate Noise Filters

Purpose: The animator creates new noise filters (i.e., Perlin Noise functions) from the menu and defining their characteristics through the noise filter editor. These filters are used to create different random number distributions that provide different and (hopefully) more interesting behaviors. One motivation for providing multiple noise filters is to drive different motion of dynamic objects or the selection of different actions. The system creates a new instance of the noise filter and sets the appropriate parameters. The system displays the characteristics of the new filter and prompts for a name. Menu options enable the animator to modify the parameters of an existing noise filter or to delete a filter. Animation sequences referencing deleted filters are mapped to an existing or default filter. The animator loads an animation sequence by selecting from a menu of sequences described by title, length, and textual or keyword descriptions. The system offers to save the current sequence,

closes the current sequence, clears the user interface and sequence display, opens the associated files, sets the related environment variables, updates the user interface, and posts a status (ready or error) to the animator.

Scenarios: Noise filters are available to every subsystem in the framework. Framework elements can use existing noise filters or the animator can create filters with specific characteristics for particular subsystems.

Fault Cases: The system cannot locate or access the desired filter. The system cannot instantiate a new filter. The requested parameter changes cause mathematical error (such as dividing by zero).

Variations: A genetic algorithm can be constructed to automatically tune and manipulate the filter parameters while searching for a "most correct" solution.

A.1.3 Manipulate Behavior Parameters

Purpose: The animator uses the menu to open the behavior editor. The system displays the behavior editor and enables the animator to specify goals, emotions, personal values, constraints and other parameters. The animator can review or update existing parameter values.

Scenarios: The animator can manipulate the behavior of any autonomous creature by altering the weighting and priority of its parameters. The Animator can also alter the behavior of non-creature elements of the animation. The animator uses the action map editor to display the current action map. The system displays a list of the current actions, input syntax, output syntax, and semantics

Fault Cases: The data entered leads to contradictory behaviors or evokes a fault during the parameter processing.

Variations: A genetic algorithm can be constructed to automatically tune and manipulate the behavior parameters while searching for a "most correct" solution.

A.1.4 Manipulate the Population Editor

Purpose: The animator uses the menu to select the population editor. The system displays the population editor window, allows the user to change population, creature characteristics, mappings to

behaviors and noise filters. The animator uses the menu to open the behavior editor. The system displays the behavior editor and enables the animator to specify goals, emotions, personal values, constraints and other parameters. The animator can review or update existing parameter values.

Scenarios: The animator sets the number and type of autonomous creatures in the animation. For each creature or set of creatures, the animator can allocate particular behaviors and assign noise filters. The animator can use the behavior or the noise filter editors to manipulate the qualities of the individual creatures.

Fault Cases: The population size is too large to create an effective a sequence. The system cannot instantiate enough creatures to meet the demand of the animator.

Variations: A genetic algorithm can be constructed to automatically tune and manipulate the population parameters while searching for a "most correct" solution.

A.1.5 Modify Execution Mode

Purpose: The animator controls the operation of the current sequence. The animator selects either a "run" or single step mode. The animator selects whether a trace of events, actions, and decision paths is displayed and/or logged. While these options are being set, the system pauses the current animation sequence, updates the user interface, removes the trace window, and resumes the sequence. In single-step mode, the system conducts one iteration, updates the trace windows and animation display, and pauses.

Scenarios: The animator resets the single step mode option on the user interface. The system resets the trace mode, resets the single step mode, removes the trace windows, updates the animation display, and resumes the sequence. The animator resets the trace mode to false. The animator selects the single step mode option on the user interface.

Fault Cases: The system cannot display the log window. The system cannot create the log file. There is no current animation sequence to execute, trace or debug.

Variations: No variations currently identified.

A.1.6 Manipulate the Environment Editor

Purpose: The animator modifies the default environment variables for a sequence. The system pauses the current sequence, provides an user interface for changing the values, the default values are saved, the new values are saved as baseline, the user interface is updated, and the current animation continued.

Scenarios: The animator modifies the current environment to include static and dynamic objects. The animator uses the editor to control the placement and characteristics of the environment elements

Fault Cases: The data leads to contradictory placement of characteristics of elements, such as a tree and a rock occupying the same space.

Variations: A genetic algorithm can be constructed to automatically tune and manipulate the environment parameters while searching for a "most correct" solution. Environment patterns can be used to quickly patch together an environment

A.2 Creature Actor Use Cases

The Creature Actor is the central character of an animation sequence. It must include a decision making authority to give it autonomy; a mobility element to give it motion; a dialogue model so that it can interact with other creatures; and a emotion model so that it believable. The relationships between the models within the Creature Actor are shown in Figure 16.

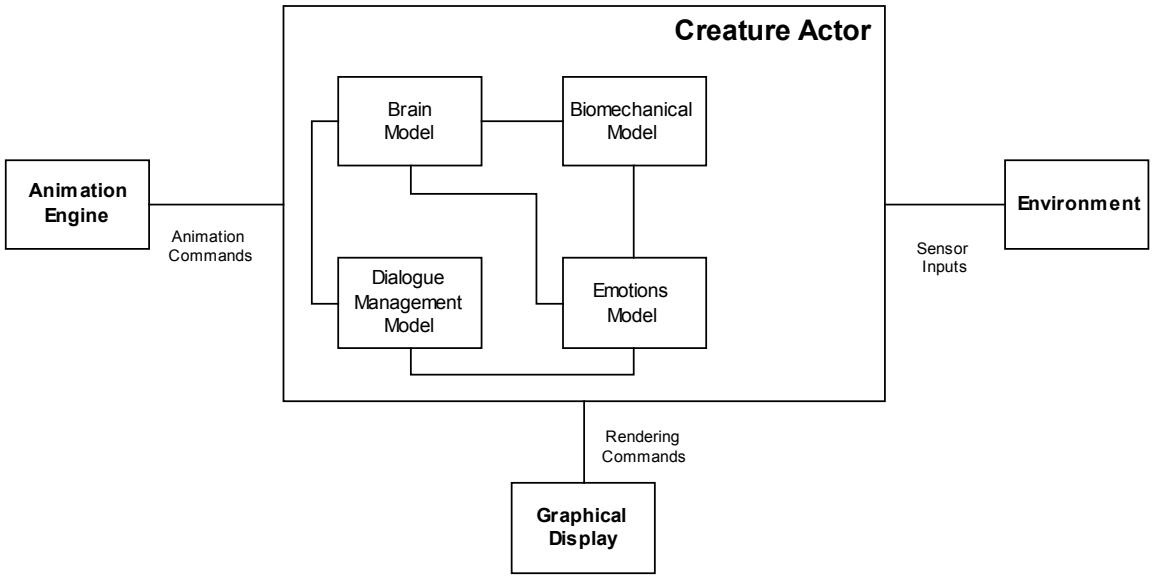

Figure 16 Relationships Between Major Sets of Creature Use Cases

A.2.1 Creature Actor: Biomechanical Model Use Case

The Biomechanical Model is responsible for the physical movement of the creature and its interaction with its physical environment. It includes models for sensory input, that is the hearing and seeing of the creature. The relationships between the elements of the Biomechanical Model are shown in Figure 17.

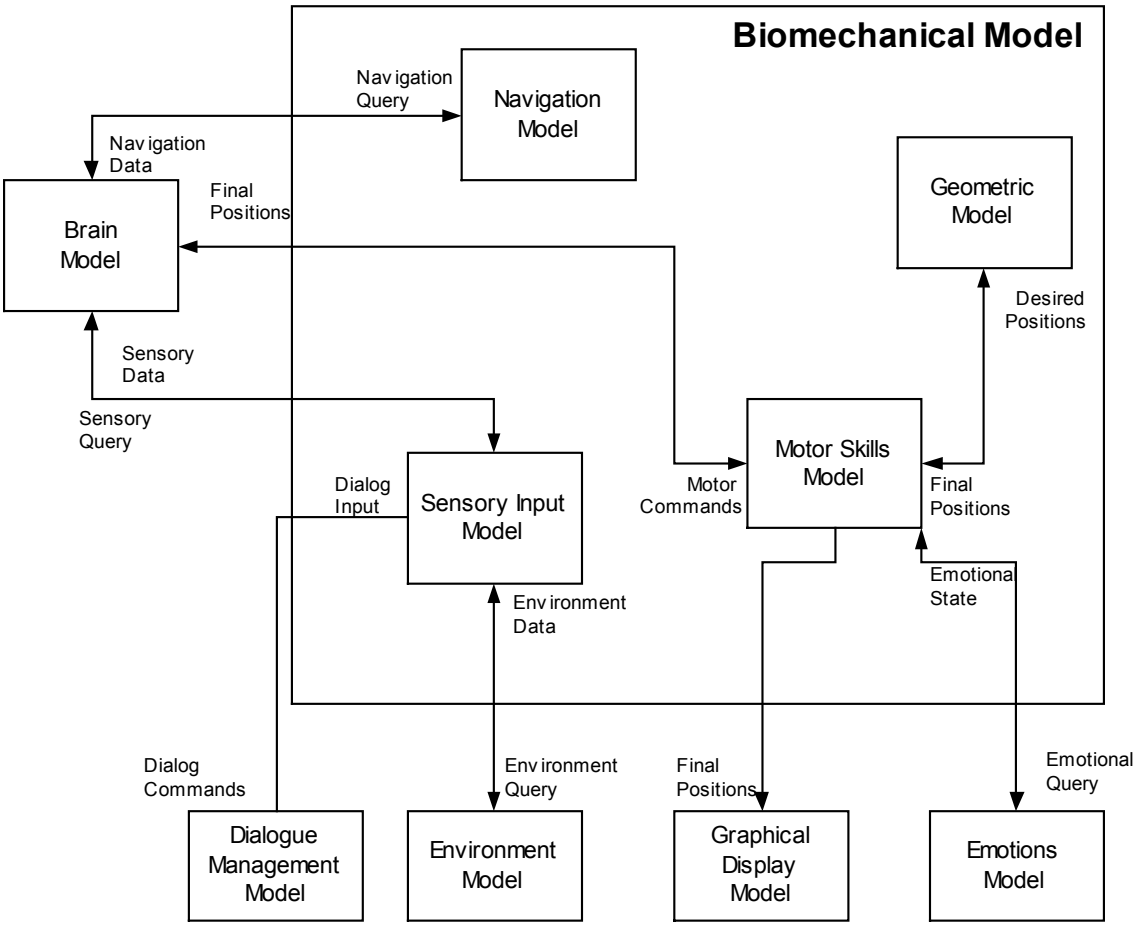

Figure 17 Relationships Between Use Cases for the Biomechanical Model

A.2.1.1 Motor skills

Purpose: Motor commands are received from the brain model to move portions of the actor anatomy. Motor skills interpret the brain commands and compute desired geometric positions and intermediary transition points of the anatomy. The Emotion Model is interrogated to determine the emotional adjectives, which alter the motor skill computations. Geometric positions are sent to Geometry Model

to validate the new positions and orientations of the anatomy and any collisions are reported back to the Motor Skills Model. Final positions are sent to the Graphical Display Model for rendering

Scenarios: Stand, Walk, Run, Jump, Swim, Fly, Facial Gesture, Body Gesture, Hand Gesture, Nod, Shake Head, Shake Body, Fidget, Push, Pull, Grab Object, Release Object, Throw Object, Catch Object

Fault Cases: Brain commands are not understood or does not match motor skills repertoire. Geometry model reports collision and brain command not possible. Brain command is only partially possible.

Variations: Motor Skill learning model is added. Alternative anatomical models are introduced including quadrupeds and disabled anatomy models. Additional motor skills are introduced. An Aging Effects Model is introduced.

A.2.1.2 Geometry

Purpose: The Geometry Model calculates the final position and oriented of anatomical elements based on the desired position and orientation received from the Motor Skills Model. The Geometry Model accounts for the physical location of objects and other characters in the environment to determine collisions.

Scenarios: Desired position and orientation do not cause any collisions and the final position and orientation can be calculated directly. Desired position or orientation violates a physical law of the environment and the final position or orientation is calculated based on the collision position.

Fault Cases: Input data or final data is nonsensical.

Variations: Variations on the geometric modeling data and degree of detail. Anatomical models (e.g., Jane Wilhelms) can be included.

A.2.1.3 Sensory Input

Purpose: The Sensory Input Model interrogates the Physical Environment Model to input data into the decision making processes in the Brain Model. The Brain Model interrogates the Sensory Model to determine the current state of the character's observable environment.

Scenarios: Visual, Audio, Aural, Taste, and Touch.

Fault Cases: The current position and orientation of the character cannot be matched to the environment.

Variations: Additional sensory models, such as extrasensory perception can be included. Diminished capacity or reduced sensory perception models, such as tired, intoxicated, illness, or disease can also be included.

A.2.1.4 Navigation

Purpose: The Navigational Model defines, monitors, plans, and re-plans paths through the physical environment. It calculates both short/local routes and longer/global routes to meet objectives from the Brain Model. The Brain Model interrogates the Navigational Model to determine short-term (i.e., next step) objectives and to contribute to the overall decision process.

Scenarios: Short-term planning; long-term planning

Fault Cases: The Navigational model cannot create short or long-term plans to meet objectives.

Variations: The Navigational Model can be extended to include contingency planning.

A.2.2 Creature Actor: Brain Model Use Cases

The Brain Model is the central element of an autonomous creature. It is responsible for sensing elements of the environment and executing actions to achieve its goals and objectives. The Brain Model includes within its decision process the effects of learning and emotion. Notice the introduction of a Learning Model in the Brain Model. The relationships between the elements of the Brain Model are shown in Figure 18.

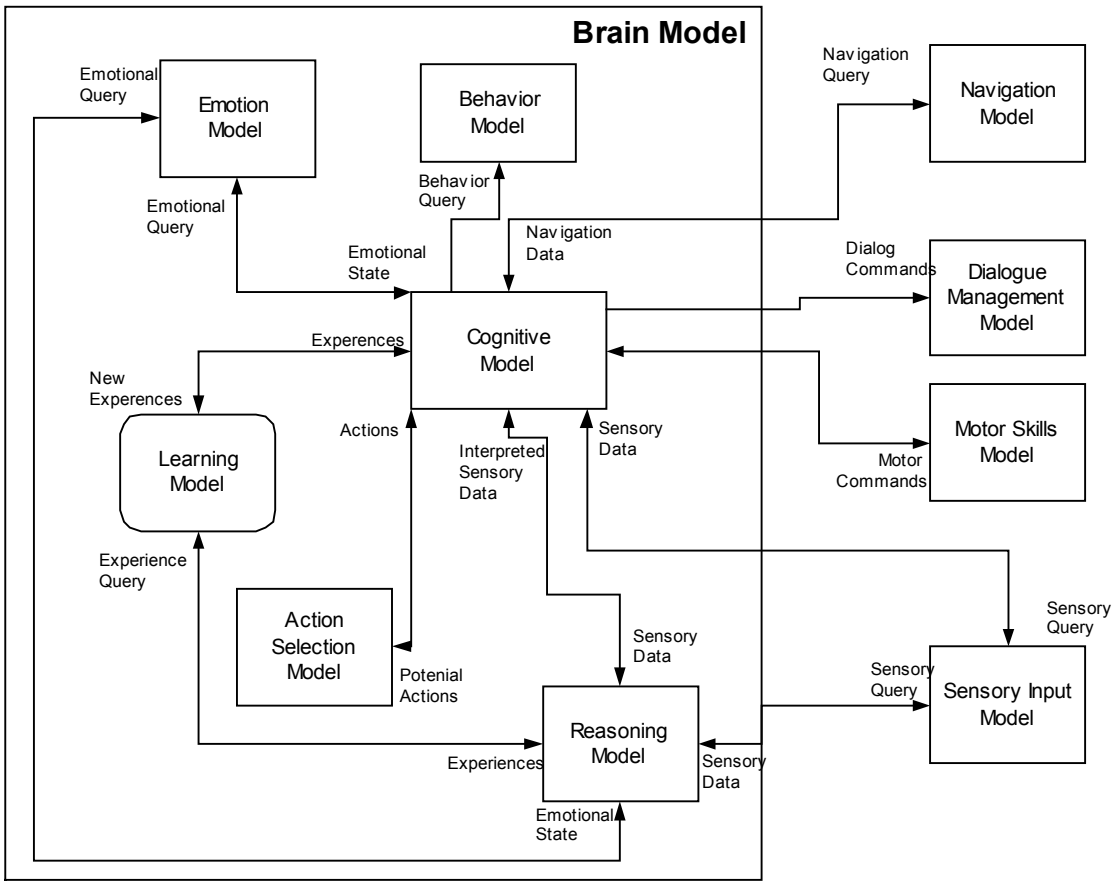

Figure 18 Relationships Between Use Cases for the Brain Model

A.2.2.1 Cognitive Model

Purpose: The Cognitive Model expresses and executes the thought and decision process to achieve character goals within behavior and environment constraints. The Cognitive Model determines how characters will react and interact with their environment, circumstances, and other characters.

Scenarios: Prioritize basic character goals based on the behavior rules and current circumstances. Prioritize tasks to determine course of action. Interrogate Sensory Model to determine current conditions and presence of unexpected events. Analyze goals, tasks, and conditions to determine potential courses of action. Interrogate learning model to determine of experience exists on how to best achieve goals given current circumstances. Update Learning Model with results from previous tasks and actions and effectiveness for the given circumstances. The Cognitive Model interrogates the Emotion model to determine the current emotion conditions and its behavior modifiers. Constraints

from the environment, behavior model, and previous bad experiences (from Learning Model) are applied to the decision process. The Cognitive Model issues potential actions to the Action Selection Model, which in turn selections the appropriate actions. The selected actions are issued as commands to the Motor Skill model, and the Dialogue Management Model. Updates on short-term and long-term plans are issued to Navigational Model and to the Emotion Model.

Fault Cases: The Cognitive model cannot correctly process an input from the sensory model. The model cannot determine a course of action based on objectives and sensory inputs. The cognitive model cannot determine how to respond or react to events in the environment or the actions of others characters.

Variations: Personalities traits, flaws, mental illness, variations, sociopath behaviors, and other alternative models can be incorporated into the Cognitive Model.

A.2.2.2 Learning Model

Purpose: The Learning Model matches potential tasks and actions for given goals and circumstances. It constructs experiences based on the outcome of circumstances and selected actions and decisions. The model weights the outcome of these experiences to be able to mimic good and bad experiences.

Scenarios: The Brain Model interrogates the Learning Model as part of the action selection, decision-making, and dialog communications. Results of the taking specific actions and making specific decisions in specific circumstances and the outcome of responses to specific utterances are recorded.

Fault Cases: The learning model can incorrectly fail to match current circumstances and situations causing (incorrectly) no experience modifications.

Variations: Any number of learning disorders can be modeled and incorporated into the framework.

A.2.2.3 Action Selection

Purpose: The Action Selection Model determines and computes which actions are possible with the available energy. The Cognitive Model sends a list of prioritized tasks to the Action Selection Model, which returns a list of the doable actions.

Scenarios: Various algorithms are used to determine, in priority order, which actions can be accomplished. Once selected, an action is allocated a percentage of available energy and its outcome is computed. The results of the Action Model are returned to the Cognitive Model.

Fault Cases: There is no energy or not enough energy to complete any of the requested actions. The Action Model does not include any of the requested actions.

Variations: Variations in energy consumption that model exhaustion, emotional stress, illness and other factors can be included.

A.2.2.4 Reasoning Architecture

Note: the reasoning architecture referred to here is from the conversational actor research. It refers to the recognition and comprehension of gestures and speech from other actors. The architecture seeks to reason the meaning of the gesture and form an appropriate response.

Purpose: The Reasoning Model attempts to recognize and interpret gestures and speech from other characters in the environment and to develop an appropriate response given the circumstance and behavior traits of the character.

Scenarios: The Cognitive Model acquires data from the Sensory Input Model. These inputs are sent to the Reasoning Model for interpretation. The Reasoning Model interrogates the Emotion Model, Sensory Input Model, and the Learning Model for a context in which to interpret these inputs. The model matches gestures or speech elements with known elements and modifies their interpretation according to the developed context. The results are return to the Cognitive Model.

Fault Cases: The Reasoning Model does not understand the gesture or speech element. The Emotion Model clouds the interpretation of the element. The Learning Model returns contrary advice on a course of action, or returns advice that violates a basic behavioral rule.

Variations: No variations currently identified.

A.2.2.5 Behavior System

Purpose: The Behavior Model represents the basic personality rules, primary goals and objectives, and constraints for the character.

Scenarios: The Cognitive Model interrogates the Behavior Model to determine basic objectives for the current set of circumstances. The Cognitive Model refines and modifies these objectives according to input data from other models. The Behavior rules are static rules defined offline.

Fault Cases: The Behavior Model does not have a basic objective for the given circumstances.

Variations: The Learning Model, given sufficient time and experiences could alter the basic behavior rules.

A.2.2.6 Scripted Agents

The Scripted Agent Model controls the simplistic behavior of dynamic elements of an animation that are not modeled are autonomous creatures. That is, they exhibit no unplanned behaviors. We have debated whether Scripted Agents are part of the larger Brain Model or whether they fit better into the Environment Model. For this iteration, we have chosen to include them in the Brain Model. The relationships between the elements of the Scripted Agents are shown in Figure 19.

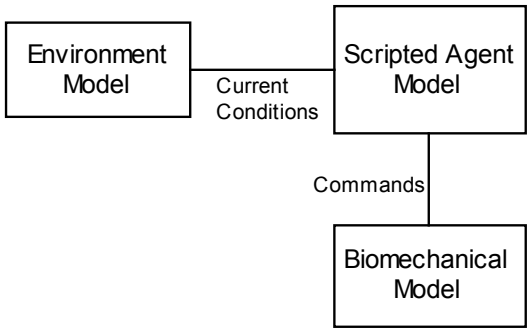

Figure 19 Relationships Between Scripted Agent Use Cases

Purpose: The Scripted Agent Model executes non-player characters according to a predefined action script.

Scenarios: The Scripted Actor Model interrogates the environment and executes the next series of actions in the appropriate script. The action commands are translated into commands for the biomechanical model.

Fault Cases: No variations currently identified.

Variations: Scripted agents could include actions input from VR devices, network connections, or other agents whose actions and cognition are controlled outside the framework.

A.2.3 Creature Actor: Dialogue Management Use Case

The Dialogue Management model is responsible the interpretation of utterances whether they come from interactively from humans or from other creatures. It is also responsible for constructing appropriate responses based on the current context and emotional state of the creature. The relationships between the elements of the Dialogue Management Model are shown in Figure 20.

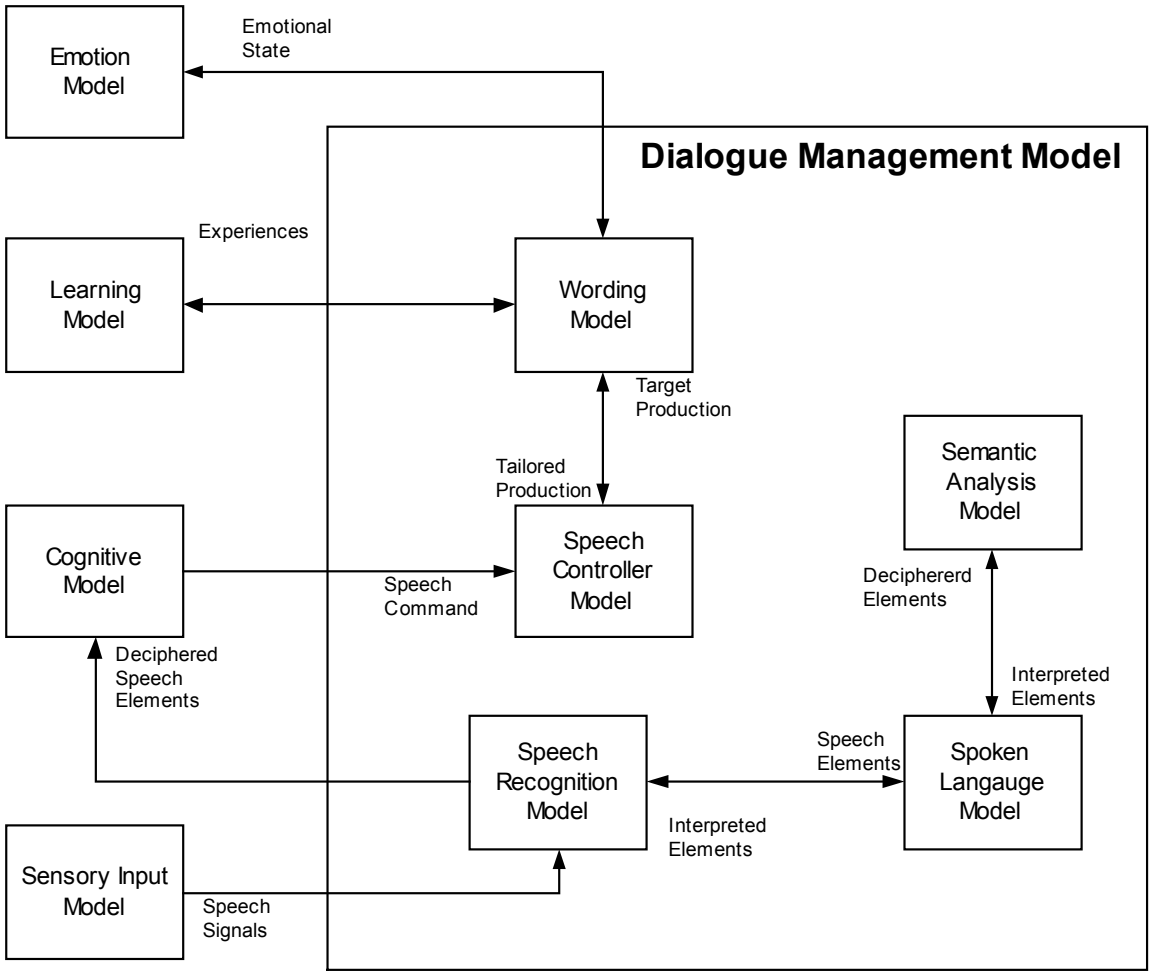

Figure 20 Relationships Between Use Cases for the Dialogue Management Model

A.2.3.1 Speech Controller

Purpose: The Speech Controller Model produces audible or electronic, inter-character speech elements. It simulates the physical act of talking.

Scenarios: The Cognitive Model issues a command to the Speech Controller Model to speak or make audible noises.

Fault Cases: The Speech Controller Model cannot articulate the requested sounds.

Variations: Several speech impediments, regional dialogs, or speech difficulties can be included in the Speech Controller Model.

A.2.3.2 Dialog or Speech Recognition

Purpose: The Speech Recognition Model parses incoming audible or electronic signals into speech elements and renders those elements into words, phrases, and noises.

Scenarios: The Speech Recognition Model interrogates the Sensory Input Model to determine if there are any detectable audible signals. The Speech Recognition Model parses any detected signals into recognized speech or sound elements. These elements are sent to the Cognitive Model as sensory.

Fault Cases: The Speech Recognition Model cannot parse or recognize all or some of input signal.

Variations: Variations on hearing or word comprehension can be included.

A.2.3.3 Wording Selection

Purpose: The Wording Model selects the desired vocabulary, phrases, words, or sounds depending on the goals, behaviors, circumstances, and emotional state of the character.

Scenarios: The Cognitive Model signals the Speech Controller Model to produce a words or sounds. The Speech Controller Model interrogates the Wording Model to tailor and adapt the given target production to account for circumstances and emotional state. The Wording Model interrogates the Emotion Model and the Learning Model to transform the requested production.

Fault Cases: No fault cases have been identified yet.

Variations: No fault cases have been identified yet.

A.2.3.4 Spoken Language Processing

Purpose: The Spoken Language Model translates the recognized speech or sound elements into commands or experiences relevant to the character.

Scenarios: The Speech Recognition Model requests a translation of a given phrase or sounds before returning it to the Cognitive Model. The Spoken Language Model interrogates the Learning and Emotion Models to help decipher the phrase or sound. The Spoken Language Model uses the results from the Learning and Emotion Models to match the given sound elements against the known elements. The Spoken Language Model queries the Semantic Analysis Model to postulate an interpretation of the given sound elements into known event, commands or actions in the environment.

Fault Cases: The Spoken Language Model cannot interpret the given sounds or speech element due to a lack of experience or command reference. The Semantic Analysis Model cannot interpret the sounds elements into a known event, command, or action.

Variations: No variations have been identified yet.

A.2.3.5 Semantic Analysis

Purpose: The Semantic Analysis Model translates known sound elements into known events, actions, or commands.

Scenarios: The Spoken Language Model queries the Semantic Analysis Model to translate a set of known sound elements into known events, commands, or actions based on input from the Learning and Emotion Models.

Fault Cases: The Semantic Analysis Model cannot translate all or part of the set of sounds into known events, actions, or commands.

Variations: Variations include physiological and narcotic impediments to comprehension, analysis, memory recall, or other aspects of semantic analysis.

A.2.4 Creature Actor: Emotions Use Cases

Emotions are a difficult and complex subject matter, but add tremendous realism to animation. The Disney book describes in great detail how emotional qualities, such as happiness and sadness add tremendous quality to Disney Animations. The following is a highly simplistic attempt to model a basic personality and emotion characteristic. Emotion and personality are very active, broad and comprehensive areas of study and research within human psychology. A proper study to support the specification and development of framework objects would require a very significant effort. The relationships between the elements of the simplistic Emotion Model are shown in Figure 21.

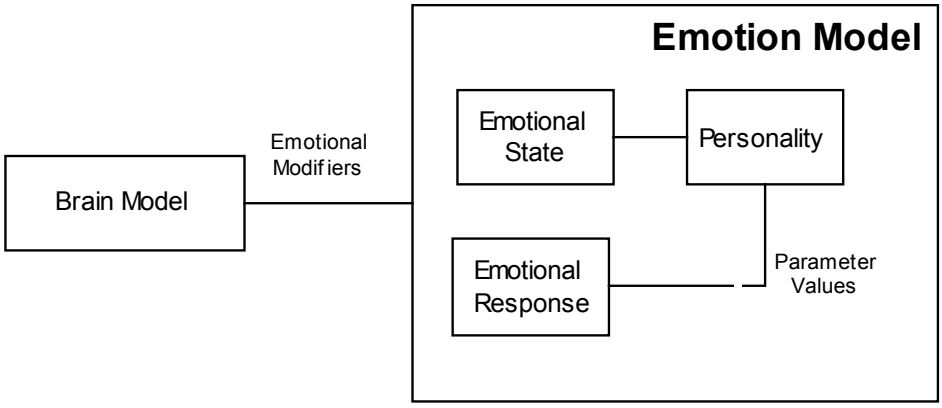

Figure 21 Relationships Between the Emotion Model Use Cases

A.2.4.1 Emotional State

Purpose: The Emotional State Model simulates the current stimuli and their effects. It models the onslaught of emotional effects and manages the deterioration of those effects over short period of times.

Scenarios: The Cognitive Model signals the results of commands, efforts to reach objectives, current events from the environment, reaction of other characters and other emotional state drivers. The Emotional State model interrogates the Personality Model to determine a foundation of the emotional state and then calculates how the effect of these inputs changes the emotional state. The Emotional State Model then sets emotional state modification parameters.

Fault Cases: Inputs create conflicting emotional states parameters (such as simultaneously being happy and sad).

Variations: Long-term emotional states, perhaps best described as moods, can be included as variations.

A.2.4.2 Personality

Purpose: The Personality Model manages a static model of the general emotional traits exhibited by a character.

Scenarios: The Emotional Response Model interrogates the Personality Model to determine the basic response to situations, events, and circumstances. The Emotional State Model also interrogates the Personality Model to as part of the calculation of the emotional state.

Fault Cases: No fault cases have been identified yet.

Variations: Multiple personalities, triggered under specific circumstances could be included in this model.

A.2.4.3 Emotional Response

Purpose: The Emotional Response Model generates modifications to planned actions or commands based on the current emotional state.

Scenarios: The Cognitive Model interrogates the Emotional Response Model to determine how planned actions or commands should be modified according to the current emotional state.

Fault Cases: No fault cases have been identified yet.

Variations: No variations have been identified yet.

A.3 Environment Actor Use Cases

The Environment Actor models and executes the physical environment for the creature or creatures. TIT contains static elements, such as buildings, trees, and mountains but also contains dynamic elements such as blowing leaves, rain, and other effects. The Environment Actor also provides and manages multiple timers to manipulate the dynamics in a sequence. The relationships between the elements of the Environment Actor are shown in Figure 22.

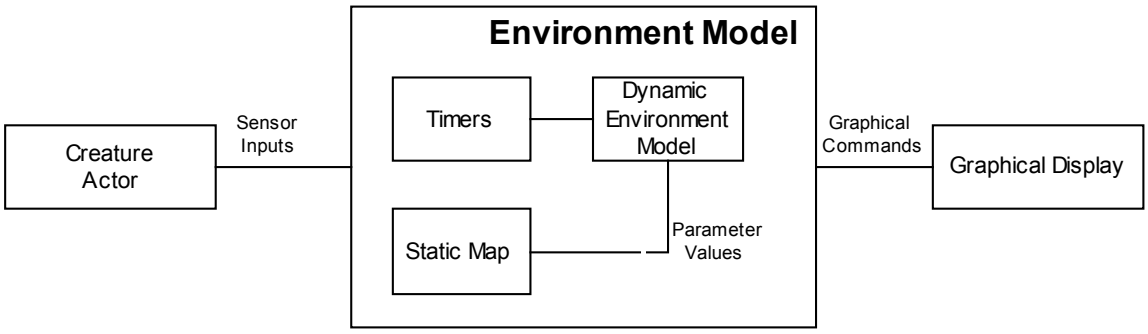

Figure 22 Relationships Between the Environment Use Cases

A.3.1 Timer Model

Purpose: The Timer Model manages the passing of time in different contexts within the environment.

Scenarios: A Model requests the creation of a timer with a given handle, reoccurrence, and duration. When activated, the timer counts from zero to the duration and halts. Any timer can be reinitialized and restarted at any point. Reoccurring timers automatically reset and restart when the duration is reached.

Fault Cases: A timer cannot be created.

Variations: A timed event could be included in the Timer model, where when the timer reaches its duration, a callback is invoked.

A.3.2 Dynamic Environment Model

Purpose: The Dynamic Environment Model manages the changing aspects of the environment and their effects on movement, sensory readings, emotions, and other character aspects.

Scenarios: The Dynamic Environment Model initializes to create timers for each dynamic element in the environment. It manages the movement and changes of these elements according to a prescribed static rule base. The Sensory Input and the Motor Skills Models interrogate the Dynamic Environment Model to determine identifiable elements, objects, events or actions and to determine collisions with other elements in the environment.

Fault Cases: No failure cases have been identified yet.

Variations: No variations have been identified yet.

A.3.3 Static Map

Purpose: The Static Map Model manages the location of permanent elements and their interaction with characters and dynamic environment elements.

Scenarios: The Sensory Input and Motor Skills Models interrogate the Static Map to determine observable elements, events, or actions and to determine collisions with elements in the Static Map Model.

Fault Cases: The Static Map Model cannot observe the environment from the given reference point.

Variations: No variations have been identified yet.

A.4 Graphical Display Model Actor Use Cases

The role of the Graphical Display Actor is to render the output of the animation processing. The Graphical Display Actor typically renders animation commands and objects into a graphical language for visual display. However, it also must be able to render information and data to files and trace windows. The relationships between the elements of the Graphical Display Actor are shown in Figure 23.

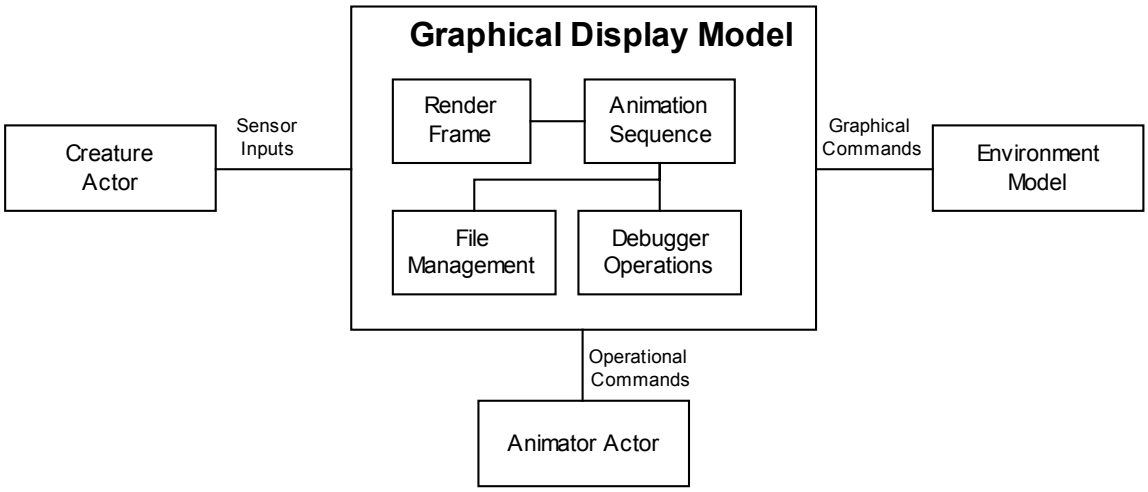

Figure 23 Relationships Between the Graphical Display Use Cases

A.4.1 Render Animation Frame

Purpose: The Render Animation operations convert graphical commands and objects into screen renderings through the graphical library calls.

Scenarios: The processing of the animation sequence derives the content of the current camera view and issues commands (and object references) to the Render Frame.

Fault Cases: The Render Frame cannot process the amount of input commands for the animation sequence step frame.

Variations: Variations include the rendering of animation sequence for multiple graphical systems such as OpenGL or DirectX.

A.4.2 Animation Sequence

Purpose: The Animation Sequence operations provide control the execution of the sequence.

Scenarios: The Animation Sequence operations increment the frame counter and control the iteration of timers, creatures, and dynamic environment elements.

Fault Cases: No faults currently identified.

Variations: The animation sequence operations provide the hot spot for extending the framework with unique capabilities for the sequence (i.e., the part of the animation that the developer customizes).

A.4.3 File Management

Purpose: The File Management operations handle the general input and output of data related to an animation sequence and any data logging required in support of sequence debugging.

Scenarios: The File Management operations respond to Animator request to load or save an animation sequence. File Management operations support the logging of data during sequence debugging.

Fault Cases: File Management operations encounter operating system file I/O errors.

Variations: No variations currently identified.

A.4.4 Debugger operations

Purpose: The Debugger Operations enable the Animator to single step through an animation sequence. Debugger Operations also enable the Animator to trace the path of execution by logging data and decision information to a trace window or log file.

Scenarios: The Animator selects the trace mode to log data and decision information to a window or file. As the animation sequence executions, key information is written out. The Animator can also select a single step mode to execute a single animation frame.

Fault Cases: The Debugger Operations encounter operating system file I/O errors.

Variations: No variations currently identified.

A.5 Animation Engine Actor Use Cases

The Animation Engine is primarily responsible for the execution of the animation. It contains the unique math routines required by the animation and provides the overall control loop. The basic relationships between elements of the Animation Engine are shown in Figure 24.

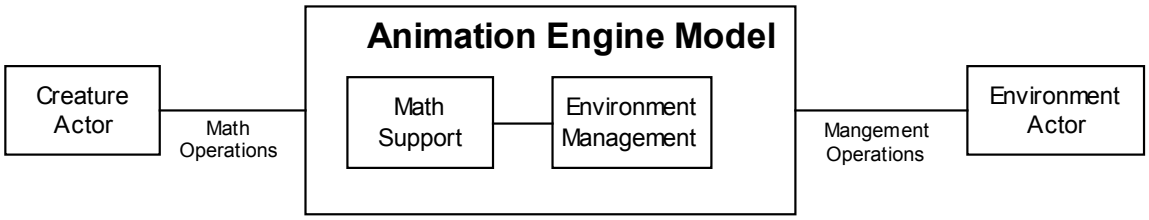

Figure 24 Relationships Between Animation Engine Use Cases

A.5.1 Mathematical Support

Purpose: The Mathematical Support operations provide a math library to support the calculation used throughout the systems. These routines include functions beyond the math library support provided by the operating system including interpolation, quaternion routines, fractal generation, filters, and other required operations.

Scenarios: Objects in the system access various math support operations to support calculations.

Fault Cases: Mathematical support operations can encounter math errors such as dividing by zero.

Variations: Numerous algorithmic variations are possible.

A.5.2 Environment Management

Purpose: The Environment Management operations provide support for defining and modifying the "business rules" for scripted agents and the static portion of the environment.

Scenarios: The Animator access the Environment Editor, which provides support for the manipulation of the environment. The Environment Editor accesses the Environment Management support operations to store the desired settings.

Fault Cases: The Environment Management operations can encounter File I/O errors.

Variations: No variations currently identified.