

**A CONSTRAINT-BASED REASONING APPROACH FOR
BEHAVIOURAL MOTION CONTROL
IN COMPUTER ANIMATION**

by

Sang Mah

B.Sc. Simon Fraser University 1985

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

© Sang Y. Mah 1993

SIMON FRASER UNIVERSITY

March 1993

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without permission of the author.

APPROVAL

Name: Sang Mah
Degree: Master of Science
Title of Thesis: A Constraint-Based Reasoning Approach for Behavioural Motion Control in Computer Animation

Examining Committee:

Chairperson: Dr. Jiawei Han

Dr. Tom Calvert
Senior Supervisor, Professor of Computing Science

Dr. Bill Havens
Supervisor, Professor of Computing Science

Dr. John Dill
Supervisor, Professor of Engineering Science

Dr. Gary Ridsdale
Supervisor, Computer Graphics Research,
MacDonald Detwiler & Associates

Dr. Dave Fracchia
External Examiner, Associate Professor of Computing Science

DATE APPROVED: _____

24 October 93

PARTIAL COPYRIGHT LICENSE

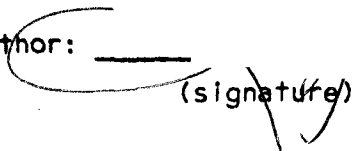
I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay. (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

A Constraint-Based Reasoning Approach for Behavioural Motion Control

in Computer Animation.

Author: _____


(signature)

Sang Yem Mah

(name)

April 14/93

(date)

ABSTRACT

In computer animation, the motion of an object is usually defined at a physical level and manipulated through mathematical models with kinematics and dynamics. The observable motion of an intelligent entity, however, is a reflection of its reasoning process as it reacts to its environment. Thus, the environment and the internal knowledge of how to handle different environmental factors serve as constraints on the motion and behaviour of an intelligent entity, and on the motion of an object controlled by intelligent entities. These constraints are qualitative and cannot be resolved by mathematical techniques for constraint-satisfaction.

In this thesis, the application of a constraint-based reasoning system in computer animation is explored. The reasoning system is used for specification and control of the behavioural motion. In particular, the proposed framework applies ECHIDNA, an object-oriented constraint-based expert system, to the animation of a sailing environment. The motion of a sailboat is controlled by intelligent agents that are responsible for plan formulation and modification, plan implementation and information extraction from the environment.

Each agent has a different set of knowledge units, or *morsels*, which impacts its task and as a result, defines individualized behaviour for a sailboat. These knowledge units describe the relationships between the sailboat and its environment (e.g. "right-of-way" relationships between sailboats), and the relationships between goals and actions. By using the ECHIDNA reasoning system to define and resolve these relationships, an alternative, model-based approach for behavioural motion control is provided for computer animation of multiple entities.

To Quasimodo
who sits alone at Jericho
while I do mental tacks, jibes and
man-overboard
manoeuvres

ACKNOWLEDGEMENTS

I would like to thank my family and friends that supported me during this quest, especially those that fed me when I vowed not to cook until the completion of this thesis!

In particular, I would like to thank my supervisor, Dr. Tom Calvert, for his support, guidance and good humor throughout this thesis work and other endeavours. I am also grateful to Miron Cuperman in the Expert Systems Lab for his help with Echidna, to Lars Wilke for his enduring support and encouragement, and to the members of the graphics research lab for making late-night work a little more cheerful.

My warmest appreciation goes to my mother, my father (in memory), and my two brothers, Douglas and Herman, for their unquestioning support of my work and concerns.

This research was funded in part by a Post-Graduate Scholarship Award and research grants from the Natural Sciences and Engineering Research Council of Canada, and by grants from the Social Science and Humanities Research Council of Canada.

TABLE OF CONTENTS

Approval	ii
Abstract	iii
Acknowledgements	v
Table of Contents	vi
List of Figures	vii
1. Introduction	
1.1 Background	1
1.2 Motivation	2
1.3 Proposed Work	3
2. Related Research	
2.1 Motion Control in Generality	5
2.2 Behavioural Animation	7
2.3 Animation of Human Behaviour	9
3. Background Information	
3.1 Problem Solving	13
3.2 Knowledge Representation	13
3.3 The ECHIDNA CLP Language	17
3.4 Classical AI Planners	18
4. Conceptual Design	
4.1 Overview	22
4.2 World Representation	24
4.3 The Reasoning Component	26
4.4 The Interface and Display Component	36
5. NSAIL Implementation Overview	
5.1 Introduction	37
5.2 The Sailing Model	38
5.3 NSAIL System Architecture	39
5.4 World Representation	40
6. NSAIL Knowledge Morsels	
6.1 Overview	45
6.2 Boat Morsels	45
6.3 Reaction Morsels	47
6.4 Plan Morsels	50
7. NSAIL Control and Animation	
7.1 Overview	54
7.2 Knowledge Agents	54
7.3 Control Hierarchy	55
7.4 The NSAIL User Interface	56
8. Discussion	
8.1 The Evolution	59
8.2 Implementation Issues	59
8.3 Representation Issues	62
8.4 Evaluation of Approach	66
9. Conclusions	70
Appendix A: General Terminology	72
Appendix B: Sailing Terminology and Model	77
References	80

LIST OF FIGURES

<i>Figure 2.1</i>	Animation of Boids	5
<i>Figure 3.1</i>	Desired Characteristics of AI Planners	19
<i>Figure 4.1a</i>	Entity and Environment	22
<i>Figure 4.1b</i>	Entities in the World	22
<i>Figure 4.2</i>	General Framework for a Constraint-Based Animation System	24
<i>Figure 4.3</i>	Physical Object Schema	25
<i>Figure 4.4.</i>	Polar Target Schema	25
<i>Figure 4.5</i>	Morsel Variable	27
<i>Figure 4.6</i>	Example Morsel	28
<i>Figure 4.7a</i>	Knowledge Agents	29
<i>Figure 4.7b</i>	The Two Sides of An Agent	29
<i>Figure 4.8</i>	The Planning Agent	30
<i>Figure 4.9</i>	Basic Planning Algorithm	31
<i>Figure 4.10</i>	Plan List Hierarchy	31
<i>Figure 4.11a</i>	Plan Node	32
<i>Figure 4.11b</i>	Relationship Between Morsels and Plan Nodes	32
<i>Figure 4.12</i>	The Implementation Agent	33
<i>Figure 4.13</i>	Basic Animation Algorithm	34
<i>Figure 4.14</i>	Goal Achievement Morsel	35
<i>Figure 5.1</i>	Having a Hobieday!	37
<i>Figure 5.2</i>	Main Sailboat Variables	38
<i>Figure 5.3</i>	Boat Symbols	38
<i>Figure 5.4</i>	Points of Sail	39
<i>Figure 5.5</i>	NSAIL System Architecture	40
<i>Figure 5.6</i>	Hobiecat Schema	40
<i>Figure 5.7</i>	BoatState Schema	41
<i>Figure 5.9</i>	Boat Heading Target	42
<i>Figure 5.10</i>	NSAIL Goal Actions	42
<i>Figure 5.11</i>	Motion Unit Structure	43
<i>Figure 5.12</i>	Tack from Marchaj64 Figure 229	44
<i>Figure 5.13</i>	Other Uses of Motion Units	44
<i>Figure 6.1</i>	NSAIL Knowledge Morsels	45
<i>Figure 6.2</i>	Boat Morsel Schema	45

<i>Figure 6.3</i>	Point Of Sail Morsel	46
<i>Figure 6.4</i>	Boat Tack	46
<i>Figure 6.5</i>	Sail Angle	46
<i>Figure 6.6</i>	Sail Angle Range	47
<i>Figure 6.7</i>	Reaction Morsels	47
<i>Figure 6.8</i>	Static Object	47
<i>Figure 6.9a</i>	Collision?	48
<i>Figure 6.9b</i>	Avoidance I	48
<i>Figure 6.9c</i>	Avoidance II	48
<i>Figure 6.10</i>	Collision Point Relationship	48
<i>Figure 6.11</i>	Land Morsel	49
<i>Figure 6.12</i>	Right of Way Rules for Sailing	49
<i>Figure 6.13</i>	Right of Way Morsel	50
<i>Figure 6.14</i>	Plan Morsels	50
<i>Figure 6.15</i>	NSAIL Plan Node	51
<i>Figure 6.16</i>	Determining Action for a Target Goal	51
<i>Figure 6.17</i>	Beating Upwind	51
<i>Figure 6.18</i>	Tack Operation Morsel	52
<i>Figure 6.19</i>	Rounding a Mark to Port	52
<i>Figure 7.1</i>	NSAIL Control Flow	54
<i>Figure 7.2</i>	NSAIL Knowledge Agents	54
<i>Figure 7.3</i>	Top Level Planning Goals	55
<i>Figure 7.4</i>	Replanning	55
<i>Figure 7.5</i>	Control Hierarchy	56
<i>Figure 7.6</i>	Screen Dump of NSAIL User Interface	57
<i>Figure 7.7</i>	NSAIL Animation Display of Hobicats	58
<i>Figure 7.8</i>	An Echidna Tack	59
<i>Figure 8.2</i>	The Element Primitive	62
<i>Figure 8.3</i>	Hierarchical Coordinate Space	62
<i>Figure 8.4</i>	Adaptive World Hierarchy	63
<i>Figure 8.5</i>	Polar Target	65
<i>Figure B.1</i>	Equilibrium Model	78
<i>Figure B.2</i>	Velocity Triangle	79
<i>Figure B.3</i>	All-Round Performance for 12 Metre Boat	79

1.0 INTRODUCTION.

Motion is the act or process of changing places, a movement from one location to another or from one configuration to another. The observable motion of an intelligent entity is a reflection of its reasoning processes as it reacts to stimuli from its environment. Thus, the environment and the internal knowledge of how to handle different environmental factors serve as constraints on the motion and behaviour of an intelligent entity, and on the motion of an inanimate object controlled by intelligent entities. Modelling the constraints or relationships between the entity and its environment empowers the animator with an alternative high-level approach to behavioural motion specification. However, these constraints are generally qualitative and cannot be easily resolved by mathematical techniques for constraint satisfaction [Brown89].

The objective of this thesis is to explore the use of a constraint-based expert system for knowledge representation and reasoning in computer animation^{1,A}. In particular, the interest is in using ECHIDNA, an object-oriented constraint reasoning system being developed at Simon Fraser University [Havens90]. ECHIDNA is the core of the reasoning component in the proposed conceptual framework for an integrated reasoning animation system. The interactive system provides high-level control of behavioural motion of multiple intelligent entities such as those found in a simulated² sailing environment.

1.1 BACKGROUND.

In computer animation, the description and manipulation of an object in motion is usually based on mathematical models or techniques. Motion is generated by methods such as interpolation through keyframes, inverse kinematics and dynamics [Calvert82, Girard85, Isaacs87, Sturman87, Wilhelms85]. This is a definition of motion that resides purely at the physical level, addressing the mechanics, or the *how* aspect, of the motion only. This level of interaction awards absolute direction of all components to the animator, but it can also be needlessly time-consuming and repetitive [Gould89, vanBaerle86].

¹Words and phrases used in this thesis which may not be considered standard terminology in the field of computing science may be found in the glossary provided in Appendix A.

²The distinction between *animation* and *simulation*, as practiced today, is unclear. Almost every "animation system" is also a "simulation system", in the traditional sense of the word (for example, [Gomez84, Girard85, Wilhelms85]). Esakov and Badler [Esakov91] distinguish an "animation system" as a system whose primary purpose is to output animation or motion whereas a "simulation system" is a system whose primary purpose is to simulate a set of tasks or actions. Ultimately, an animation system may use simulations to specify motion and a simulation system may use animation to output its results. For purpose of this thesis, the intent is to develop an animation tool or system to eventually control an underlying motion simulation system.

Goal-directed animation systems strive towards user dictation of motion in natural language commands [Badler92, Drewery86, Korein82]. This is a reference to the *what* aspect of movement specification. The animator states what action the object should perform, and the system will automatically create the appropriate output. Goal-directed systems, based on the director/actor concept, were introduced in artificial intelligence for task planning of a robot arm in a blocks world [Winograd72]. Goal-directed animation systems [Bruderlin88, Phillips88, Zeltzer84] are based on the idea that objects can take, or *learn*, assigned motor skills and be directed to use those skills. For example, the cue of "walk to the door" would initiate animation of a figure walking to the door without further input from the animator. Once the action is in progress, the object must also *know* how to react to changes in its environment. In other words, it must know how to behave.

Behaviour is defined as the response of an individual, group or species to its environment. Behaviour can be solely reactive, a reflexive response to stimulus from an object's environment, or it can be an intelligent response driven by an object's internal desires and experience. For example, the motion of an automobile, at the mechanical level, can be expressed by the physics governing bodies in motion, but the physics does not explain why a car turns right or left. The car, like many objects in the world, is controlled by an intelligent entity whose reasoning process must be represented. This is the *why* aspect of motion. Behavioural animation systems [Amkraut85, Reeves83, Renault90, Reynolds87, Wilhelms90] define the conditions in which an action should be selected. Such conditions can be structured in a knowledge-based system as production rules with an "IF <condition> THEN <action>" template [Morawetz89, Ridsdale86], or as another representational forms [Cerccone87].

1.2 MOTIVATION.

Behavioural motion is the consequence of the reasoning process of intelligent objects as they respond to external stimuli from their environment and their internal goals. The kinds of problem solving performed by intelligent objects are conducive to approaches that are declarative and involve heuristically guided searching through a solution space. The solutions to these kinds of problems cannot be easily computed. They are determined by refinement of the solution space with constraints imposed by internal and external factors [Brown89]. Thus, it is advantageous to investigate the use of a declarative approach for animation of behavioural motion.

The ECHIDNA constraint-based reasoning system provides powerful and efficient constraint resolution [Havens91]. A constraint logic programming (CLP) language offers a wider representation range and the efficiency of specialized programs, in addition to the nondeterminism, relational form and declarative semantics of logic programming [Clocksin81, VanHentenryck89]. With nondeterminism inherent in the language, the system developer does not have to program tree searching algorithms. The relational form facilitates the representation of constraints since constraints are basically relationships. Specialized domain-dependent programs improve search facilities with the inclusion of heuristics related to the character of the domain. Declarative semantics connote flexibility since it is easier to modify and extend declarative knowledge than procedural knowledge [Gensereth87]. An object-oriented paradigm supports diversity, reuse and structure in knowledge base development [Havens92]. Integrating the ECHIDNA reasoning system into a modular animation interface provides a high-level object-oriented framework for behavioural animation.

Animation of a sailing environment provides a number of interesting characteristics: 1) multiple objects with similar behaviour; 2) multiple levels of control; 3) multiple levels of knowledge and reasoning (i.e. planning and reactive); and 4) interaction with environmental factors and with other objects. The *behaviour* of a sailboat, which is visible to people watching from shore, is defined not only by the physical mechanics of a wind-powered device in response to changing environmental conditions, but also by the character of the intelligent agent(s) controlling the physical device. Defining the profile or the knowledge of these agents will have a great impact on the resulting animation. The interest of this thesis is to model not the underlying physical layer of a sailboat, but the knowledge and decision-making process of the intelligent agents controlling the sailboat under different environmental conditions amidst other agent-controlled sailboats. Animating a complex domain by manipulating the environment and the declarative knowledge of the associated agents is a challenging problem [Calvert88, Ridsdale87, Thalmann88], and provides a novel high-level technique for modelling and manipulating behavioural motion.

1.3 PROPOSED WORK.

The primary objectives of this thesis are to: 1) explore the use of the ECHIDNA constraint-based reasoning system for behavioural motion control in computer animation; 2) propose a high-level, object-oriented approach to the animation of multiple, intelligent objects where defining the environment and the behaviour of a single object instance

automates motion generation; 3) apply the proposed approach to the implementation of a sailing interface; and 4) investigate the potential for a generic reasoning tool for animation of other domains.

The proposed structure for the constraint-based animation system for multiple objects consists of three main components: 1) planning; 2) implementation of the plan; and 3) perception-driven modification of behaviour and the plan. Each object, such as a sailboat, will have agents representing each of the three components. What differs between the objects is the contents of their knowledge base(s) used for planning and implementation of the plan. The behaviour of the object is thus defined by the *morsels* of knowledge possessed by its agents. The final animation is dependent on specification of the *knowledge profile* of objects and the description of the environment being animated. Thus, control of an animation is done at a much higher level.

The next chapters in this thesis will expand on this proposed approach for a constraint-based animation system for multiple intelligent objects. Chapter two is a review of related research, identifying past and current work involving a behavioural approach to computer animation. Chapter three provides background information on knowledge representation. In chapter four, the conceptual design for an integrated reasoning animation system evolves. Chapters five, six and seven present implementation details for the NSAIL implementation built with the proposed framework. Chapter eight is an evaluation of the proposed approach, acknowledging successes and difficulties in the design. Finally, chapter nine presents the conclusions drawn from this thesis research.

2.0 RELATED RESEARCH.

Animation of behavioural motion is an evolving area of interest in computer graphics research, recently attracting the attention of film makers [Levy92b]. The objective of behavioural animation is to create the motion of an aggregate by defining the behaviour pattern of a prototypical member. The whole motion, like that of a gestalt, is more than simply the sum of the parts. The motion of the aggregate has its own flow and character that is intriguingly different from the motion of a single member of the aggregate .

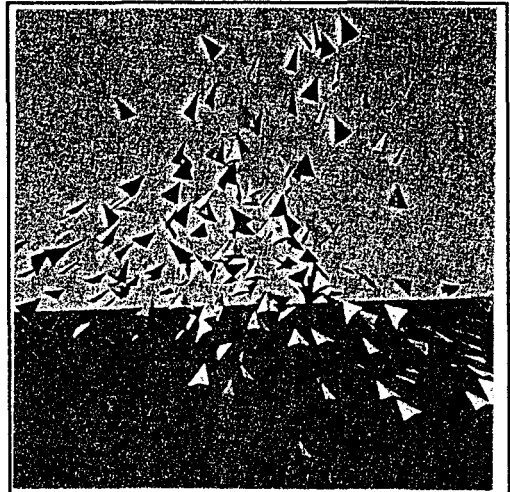


Figure 2.1 Animation of Boids

Since the members possess "intelligence", the ability to learn, understand or deal with new or trying situations, research in behavioural animation has naturally lead to an exploration of artificial intelligence (AI) techniques [Calvert88, Ridsdale90, Thalmann88, Wilhelms87]. One particular area of interest is knowledge-based frameworks for human figure animation [Badler89, Esakov91, Morawetz89, Ridsdale87, Thalmanns86, Zeltzer83]. In such systems, the primary concern is the representation of knowledge and the reasoning process, regardless of whether the intelligent entity is controlling the motion of its own body or that of another physical object. Stimuli from the environment provokes the entity into action, in a reflexive response or as a conscious response to satisfy a desired goal. This requires some planning ability by the entity to achieve the goal [Ridsdale87, Zeltzer83], and the ability to modify or redo the plan during its execution.

2.1 MOTION CONTROL IN GENERALITY.

An animator usually controls the motion of objects by explicitly specifying the frame to frame changes needed to create the illusion of movement [Sturman86]. The rate of display is set at 24 frames per second for film (or 30 frames per second for video) so that the human eye will perceive continuous motion. This leads quickly to a large number of frames that must be created and rendered in computer animation systems.

The oldest, and most widely used, method for reducing the number of frames is keyframing [Burtnyk71]. Keyframing is the basic approach taken in commercial 3D

animation systems such as Vertigo [Vertigo], Alias [Alias], etc. In a keyframing approach, only the frames in which there is a key change in motion are specified. The difference from keyframe to keyframe should be significant, but not so great that meaningful interpolation is impossible. Depending on the complexity of the movement, keyframes can be separated by five or more inbetween frames. The 2D or 3D animation system then interpolates the inbetween frames using linear, quadratic or cubic splines [Catmull78, Kochanek84, Sturman86]. Thus, in keyframe animation systems, the animator needs only to specify the positions at keyframes representing a change in the motion. However, this is still a time-consuming task: each keyframe can be quite intricate; the number of keyframes is still high; and the level of interaction is still low for complex motion specification.

Parameterized keyframing provides a slightly higher level of interaction for motion control [Hanrahan85]. Instead of specifying motion by detailing the positional changes from keyframe to keyframe over time, the animator interactively manipulates parameters for objects and motions. Restrictions on an object's motion are defined implicitly within the system. Interpolation of the parameters creates the inbetween frames. This approach is applied in various areas of figure animation including facial animation [Parke82, Thalmann88, Zeltzer85]. Bruderlin [Bruderlin93] extends the parameterized keyframing approach to the animation of human walking. The walk animator specifies values for parameters such as step size, pelvic list, etc. to define individualized gaits (some of which are very amusing) for a walking human figure.

Other automated approaches include kinematics^A [Calvert82], inverse kinematics^A [Badler87, Forsey88, Girard85, Korein82], dynamics^A or physically-based simulations [Bruderlin89, Isaacs87, Wilhelms87]. These approaches are still relatively low level where the animator must control the motion by manipulating a single joint or degree of freedom. Dynamics require the user to input values for forces and torques, or masses and springs from which forces and torques are calculated. These values do not create an intuitive platform to view or describe motion. However, the motion automatically generated from dynamics simulation is based on the laws of physics, and thus, are very realistic. This sense of realism has to be otherwise captured manually by the keyframe animator. Researchers are presently looking at better estimation techniques for dynamic parameters and better interfaces for specifying those parameters [Issacs87, Kass92, Wilhelms89].

Kinematics and dynamics may be combined for better control [Bruderlin90]. Kinematics and dynamics may also be combined with constraints, or constraints may be

used alone, for an alternative motion generation technique³. In physically-based simulations, the motion of an object is completely determined by its initial position and velocity, and the forces applied along the course. Thus, the animator must provide the right *initial* values to move to the *final* state. Control is improved by constraining the final position as well as the initial, creating a two-point boundary problem that requires more elaborate computations [Gill81]. Constraint force methods [Barzel87, Issacs87] define *positional* constraints (e.g. constraints on the positions of joints in an articulated body [Badler87]) to ensure an object moves along a predefined keyframed trajectory in the world space. For example, positional constraints ensure that a character's feet move along a given path. This reduces the amount of information that needs to be input, but is likened to dragging a character around like a marionette [Kass92]. There is no consideration of how the movement and the controlling forces change over time (i.e. the trajectories must still be determined manually).

Spacetime constraints [Kass92, Witkins88] adds another layer of control between the animator and a physically-based model of motion. The spacetime constraint approach solves for an object's motion over the *entire time interval* of interest, rather than sequentially through time as with positional constraints. This determines the necessary trajectories for the dynamic simulation. Constraints that are placed on the initial, intermediate and final positions and velocities of the object encode the goals of the motion over time. An interactive graphical interface is provided to develop complex behaviour by joining *function boxes* that hide the underlying dynamic equations [Kass92]. In this manner, the animator can control, at a higher level of interaction, dynamically-based motion by articulating the desired characteristics of the motion (e.g. quickly, gracefully, etc.).

2.2 BEHAVIOURAL ANIMATION.

Behavioural animation, a term first introduced by Reynolds [Reynolds87], also explores higher levels of motion control for an underlying simulation or animation system. In computer graphics, behavioural animation refers to the animation of a group by defining the behavioural rules for a prototype member. The animator does not need to animate each entity individually, which can be a very challenging feat when the population to be animated is in the hundreds. In traditional approaches, and in currently available

³Note that constraints in this context refer to numerical constraints or *equations* that can be solved by standard numerical optimization techniques. In a constraint-based reasoning system, constraints are *relationships* (which may be equality) and cannot be solved generally by numerical techniques.

commercial keyframe systems, this becomes a very exhaustive and rigid approach. Making even the smallest of changes becomes a horrendous task. Behavioural animation approaches control the animation of the population by defining the behaviour of its members.

Particle systems [Reeves83] are collections of large populations of individual particles, each with its own behaviour. During a particle's limited lifespan, they perform simple behaviours that alter their own internal state, consisting of colour, opacity, location and velocity. Particle systems have been used to model fire, smoke, clouds and waves .

Reynolds' object-oriented, distributed, scripted flocking system [Reynolds87] is similar to a particle system. However, the small dot-like particles are replaced with 3D geometric objects with orientation. The bird-like entities, or *boids*, in the flock exhibit more complex, and *interactive*, behaviour. Where each particle has only an internal state and behaves independently of other particles, each boid has an external state as well and must interact with other boids in order to maintain correct flocking behaviour. Each boid has three behavioural rules: 1) avoid collisions with nearby flock mates; 2) attempt to match velocity with nearby flock mates; and 3) attempt to stay close to nearby flock mates. These rules are known respectively as the collision avoidance rule, the velocity matching rule and the flock centering rule. There is actually a fourth rule, the migratory rule, which states that a boid must fly towards a given global target. These rules are processed by the boid brain.

The boid brain is divided into three separate modules: navigational, pilot and flight. The navigational module is responsible for combining, prioritizing and arbitrating between conflicting rules. A weighted averaging technique is used to combine the strength of an *acceleration request* from each rule that determines the direction and velocity of a boid. This causes incorrect behaviour in critical situations when two opposing requests cancel each other (i.e. the boid flies into a brick wall). For example, the collision avoidance rule and the flock centering rule may dictate opposing directions for flight. Although "techniques from artificial intelligence, such as expert systems, can be used to arbitrate conflicting opinions" [Reynolds87], Reynolds employs a priority ordering scheme on the acceleration requests. The priority ordering scheme assigns priority values to the rules for different situations. A single acceleration request is then given by the pilot module to the flight module which guides the flight motion of the boid. During flight, collision avoidance follows the steer-to-avoid approach rather than the force field approach used in a previous flock animation [Amkraut85] and in other animation systems [Ridsdale87]. The object-

oriented design of the system is based on the actor model for distributed systems [Agha86, Hewitt77] which has been employed in other animation systems [Reynolds82, Thalmann85]. The modularity of the system design is intended to facilitate expansion of the flock model to explore *mental states*, and plug in animation sequences created by a *real* animator [Reynolds87]. A mental state is a term referring to more complex motivations (e.g. find food, avoid predators, etc.) of existing behavioural models [Braitenberg84].

Behavioural animation also includes stimulus-response animation [Lethebridge89, Wilhelms89, Wilhelms90]. Wilhelms [Wilhelms89] explores an interactive approach to behavioural animation based on Braitenberg's *Vehicles*, a book on his work in neurobiology [Braitenberg84]. An animator controls the animation by mapping sensors and effectors of objects in a connectionist network. Sensors detect specific characteristics of objects in the environment. Effectors produce some change in the object or a force that causes motion. These forces feed into a simple underlying dynamic simulation [Wilhelms88] to produce the motion of the object. The nodes that connect the sensors and effectors have particular *transfer functions* such as love and hate. The nodes quickly became parameterized to provide a higher level of control to the user [Wilhelms90]. In this manner, the relationship between objects can be defined, animated and studied.

Other behaviourally controlled animations include Sims' thesis work in animating the locomotion of jointed objects, such as spaceships, inchworms and quadrupeds, over complex terrain [Sims87]. Research in using *biological or evolutionary* simulation for animation has gained renewed interest. *Artificial life* [Langton89, Levy92a] controls animation of populations by modelling simple local rules extracted from biological behaviour. This area includes work with botanical L-Systems [Prusinkiewicz90], digital Darwinism [Sims91], PolyWorld's artificial ecology [Yaeger82], and flocking behaviour [Reynolds87]. In addition, Haumann [Haumann88] describes a testbed for behavioural simulation of flexible objects. Morawetz [Morawetz90] explores an approach for animating human secondary movement where behaviour is dependent on user-specified personality traits (i.e. nervousness, aggression, etc.). Also, Renault [Renault90] proposes a vision-based approach, based on Reynolds' boids, to human behavioural walking animation.

2.3 ANIMATION OF HUMAN BEHAVIOUR.

Behaviour is the manner in which living organisms react to stimuli from their environment. The way a person or a personified object behaves reflects its character and

intent [Schefflen72]. There is some rationale behind the behaviour, and motion is the externalization of that rationale. Behaviour has an emotional, cultural and social context. When an object moves, its movement may be perceived as aggressive, graceful or amusing. People can attach emotions to the most simple of objects. When an object moves, it can also indicate its intention to avoid a deep hole, an angry dog or other obstacles in its surroundings. If the surroundings should change, the motion may also change. Modelling human behaviour, or *intelligent* behaviour, from this contextual level seems naturally related to work in AI. Integrating formalisms for knowledge representation and reasoning, the control of human behavioural animation advances to a higher level of interaction. The animator can conceptualize motion in terms of its behavioural characteristics rather than positional values. Techniques specialized for human movement and behaviour specification can also be classified according to the level of user interaction, ranging from guiding tools to program level tools to task level tools [Zeltzer85] and to behavioural level tools.

Guiding level tools for specification of human movement include keyframing systems such as LifeForms [Calvert93, Schiphorst90]. In keyframing systems, the animator defines the behaviour of the animated character by creating keyframes through direct manipulation of joints in an articulated body. The system then interpolates through the keyframes to produce the movement. The advantage of keyframing systems is that the systems are easy to learn, the animator has full control of the motion, and the animator draws on creative skills to choreograph the desired behaviour. The disadvantage is that the animator has full control, and is responsible for exactly *how* the motion proceeds at all levels [Kass92]. This is a disadvantage for especially complicated motion or scenes [Zeltzer85]. Other guiding tools add recording equipment or procedures (e.g. rotoscoping, puppets, electrogoniometers) to capture actual human movement [Abel85, Ginsberg83, Calvert82]. Shape interpolation [Gomez85] is the 3D equivalent of 2D keyframing. Key transformation [Gomez85] is the application of parameterized keyframing to transformation hierarchies for articulated bodies.

Program level, or animator level, tools refer to the use of special procedures or programming languages for animation. Animation details are recorded in a script for automatic generation of motion. Scripting is found in numerous human figure animation systems [Maiocchi90, Ridsdale90]. Notation-based systems [Calvert82] for movement composition may also be considered a form of scripting. Algorithmic specification systems such as GRAMPS [ODonnel81] and MIRA [Magnenat86] provide data or functional

abstraction facilities. In GRAMPS, movement at the joints can be constrained to values input from a dial guiding mechanism. MIRA is a programming paradigm closely related to the object-oriented system used in ASAS [Reynolds82]. TEMPUS [Badler82], restricted to positioning and orienting human figures, controls motion interactively through simulated potentiometers. These program-level tools enhance motion control at the *joint-level*.

Task-level tools support the animation of intelligent activity. To quote Boden [Boden77]: "Intelligent action is an act or decision that is goal-oriented, arrived at by an understandable chain of symbolic analysis and reasoning steps, and is one in which knowledge of the world informs and guides the reasoning." Task-level tools incorporate knowledge and reasoning into the animation process so that the animator can specify *what* goal or task is desired and the tool determines *how*. Goals may be articulated through natural language [Badler90, Badler92, Webber92], scripts [Morawetz89, Ridsdale90] or interactively entered goal values [Bruderlin93]. Goal-directed human behaviour animation systems include JACK [Badler89, Lee89], GESTURE [Morawetz89], GAITOR [Bruderlin93]. Other goal-oriented systems also address knowledge-based animation of complete environments for intelligent behaviour [Ridsdale90, Thalmann88, Zeltzer83].

An early knowledge-based framework for animation of articulated figures emphasized the following features: a motion control hierarchy from mechanical to behavioral; separate data and knowledge bases; blackboards^A, scripts^A and frames for knowledge representation; planning and goal-directed motion specification [Zeltzer83]. In this framework, the object-oriented *frame*^A data structure, first introduced by Minsky [Minsky75], provides the knowledge representation form. A frame is a type of schema^A used to describe a collection of attributes for a given object [Rich83]. Each *slot* in a frame, identifying a characteristic of the object, may be filled by other frames describing these objects. Procedural information may also be associated with a particular slot [Fikes85]. Zeltzer proposed a frame organization for motion, where the slots represent information attached to a skill, movement function or preconditions for execution of a skill. Frames have been applied similarly in other animation frameworks [Drewery86, Thalmann86].

Thalmann [Thalmann88] describes characteristics of an *attribute-based* system which adopts the frame-based approach. In an *implicit* animation system, the behaviour of natural phenomena would be inherent in the system (e.g. wind, temperature, sun, gravity). The objects would have *natural attributes* that are accessed by the system to automatically calculate the motion of objects under natural forces. This implies generalizing a physically-

based modelling to the extent that it can be included in such a system. It is not clear how such a generic, all-encompassing system can be implemented, but the complexity due to multiple and mutually-dependent attributes is acknowledged. Initial work includes EXPERTMIRA [Thalmann86], a Prolog-based tool for image synthesis and animation which automates control of lighting and camera movement. Research continues into the development of an integrated system for *synthetic actors* [Thalmann88, Thalmann92].

Ridsdale [Ridsdale86] applied a rule-based expert system approach to animation of human figures from a given *film script*. The script, describing the actors and their actions on stage, is translated by hand into a form that is understood by the inference engine within the program. The declarative knowledge base consists of rules from theatre and film direction, as well as scene-specific knowledge about characters and the environment. Direction rules include rules about actor placement and movement direction (e.g. do not walk in front of the speaking actor). Relational constraints define simple weighted relationships between actors and objects in the scene (e.g. attraction and repulsion). The rules, character relationships and script are used to generate scene action. A hierarchical planner, similar to the task manager in [Zeltzer82] and path finding techniques from robotics [Brooks83, Lozano83], resolves the actual paths taken by the walking figures. The planner adopts a *divide-and-conquer* strategy where the top-level problem (e.g. reach a particular destination) is broken in to an abstraction hierarchy of smaller subproblems (e.g. walk around a table). The higher, more general problems in the hierarchy are solved before the detailed subproblems.

Ridsdale's system, *The Director's Apprentice*, was later extended to investigate representation of theatre and film direction knowledge with a logic programming approach [Ridsdale87]. A more recent version [Ridsdale90] is implemented with an object-oriented extension to LISP called FROBS [Muehle87] which incorporates frames as the knowledge representation formalism. Other research that integrates film direction in animation includes a system for automated presentations in user interfaces and instruction systems [Karp90, Karp93]. Current work in comprehensive animation environments strives toward the creation of virtual *microworlds* [Brett89, Zeltzer85].

3.0 BACKGROUND INFORMATION.

3.1 PROBLEM SOLVING

Intelligent behaviour reflects a special kind of computation that is discrete, symbolic⁴ and qualitative. It is characterized by a collection of general strategies that minimize the inherent complexity of the computation. Intelligent behaviour, such as diagnosis, design or sailing, may be represented as a problem to be solved by a quantitative computation technique, but the underlying solution space would have an explosively large number of parameters, and the solution algorithm would be *NP*-complete (i.e. no general and efficient algorithm exists to solve the problem) [VanHentenryck89]. These types of computations need to be solved by a class of methods that are "intelligent" in the sense that they explore a problem space, that has been implicitly defined by a problem representation, with generic search strategies that exploit qualitative heuristic knowledge about the problem domain [Brown89]. The essence of *The Problem Space Hypothesis* [Newell80] is that humans engage in this form of problem solving.

3.2 KNOWLEDGE REPRESENTATION.

In modelling intelligent behaviour, the key aspect is knowledge representation. Many diverse kinds of knowledge are required, including knowledge about objects in the world, knowledge about processes, and *commonsense* knowledge about goals, motivation, causality, time, actions, etc. There are various formalisms in use today for representation of knowledge: logics (first order logic is especially important), semantic networks, procedural representations, logic programming, frame-based structures, production system architectures and knowledge representation languages. Knowledge representation is an important part of AI research because of the *frame problem*⁵ [McCarthy69], which is "the problem of maintaining an appropriate informational context, or frame of reference, at each stage during the problem solving process" [Cercone86]. Overviews of knowledge representation for the specialist and the naive may be found in various articles and texts [Brachman85, Cercone87]. For the purpose of this thesis, formalisms applicable to the development of the proposed integrated reasoning and animation system will be discussed.

⁴From [Rich83]: The heart of AI research is based on *The Physical Symbol System Hypothesis* [Newell76] which states that a physical symbol system has the necessary and sufficient means for general intelligent action. This provides a significant theory of the nature of human intelligence, and is the basis of the belief that it is possible to develop programs that can perform tasks done by people that are deemed intelligent.

3.2.1 Logic

First-order logic evolved from early work in theorem proving [Nilsson71] and has been recast into more computationally oriented frameworks including the Planner formalism [Hewitt72], Strips planning paradigm [Fikes85], and the widely used Prolog [Colmerauer73] programming language. Concerns about Prolog include the lack of an explicit scheme to encode knowledge, the poor handling of change and incomplete knowledge, and the perceived limitations of deductive inference. However, the formal precision and interpretability of the language permits an expressiveness that other knowledge representation schemes lack [Cerccone86].

Prolog is a "descriptive" as well as "prescriptive" language, viewing problems in terms of objects and formal relationships between objects. It asks which relationships are "true" in the solution rather than "prescribing" the sequences of steps to the solution. It is based on a standard backtracking search which is depth-first with chronological backtracking (i.e. going back to the first choice point with an untried clause). However, any search strategy may be constructed in a Prolog program (i.e. generate and test, forward checking, looking ahead, etc.). Prolog is well-suited for implementing *intelligent* programs because it can handle nondeterminism, parallelism and pattern-directed procedure call. These notions are the core of intelligent programs [Clocksin81]. Descriptive⁶, or declarative, forms are also easy to modify and extend since they involve only changes to the *data* rather than the *program* [Rich83].

Akin to the belief that people possess *procedures* for the variety of actions that can be performed, procedural knowledge is a representation that views knowledge in terms of how it is used. Knowledge embedded in procedures is accessed directly or through pattern directed procedure invocation [Hewitt72]. The procedural perspective has been most influentially championed by Winograd's work with SHRDLU [Winograd72]. Limiting the domain to a microworld of blocks, hence known as *the blocks world*, the SHRDLU system applied procedures to interpret and generate a wide variety of natural language sentences that pertained to the stacking and unstacking of blocks. Given a goal (e.g. pick up red block and put in blue box), the system determines the sequence of actions to achieve the goal. Winograd chose the MICROPLANNER programming language to implement the

⁵Cerccone [Cerccone86] offers an amusing illustration of the frame problem with a story about three robots (R1D1, R2D1 and R2D2) and three bombs.

⁶Note that the comparison of descriptive versus prescriptive languages uses the same conceptual argument, but different terminology, as the comparison of declarative and procedural knowledge [Nilsson71].

interpretation procedures. MICROPLANNER is based on a subset of the PLANNER formalism from first order logic. MICROPLANNER fell into disuse when its "blind backtracking" search could not be controlled by its user. Prolog tries to alleviate this problem with its CUT operator (see Clocksin81), which is used to indicate which previous choices should not be considered again when backtracking. However, if too many CUTs are employed to make a Prolog program execute in real time, the program becomes opaque and intractable [Cercione87].

The classification of knowledge representation frameworks (e.g. logical versus procedural paradigms) is not clearly defined or mutually exclusive. In fact, representation frameworks often influence each other and may be integrated for greater expressiveness.

3.2.2 Logic Programming

Logic programming (LP) is the combination of logic and procedural representations of knowledge, which has been the basis of *logic programs* for automated reasoning and natural language understanding [Hadley85]. The power of logic programming lies in the fact that its declarative sentences in predicate calculus^A are also programs [Cercione87], which provides a procedural formalism with strong semantic foundation [Lloyd84]. Simple logic programming examples are given in the introduction of various LP books and manuals [Clocksin81, Sidebottom92b, Sterling86] and in general AI texts [Rich83].

Logic programming, with its nondeterminism, relational form and declarative semantics, is well-suited for solving discrete combinatorial problems such as scheduling, scene labelling, operations research, etc. Problems in this class are mostly *NP*-complete, and are generally solved by searching in a finite discrete space for a point satisfying a set of constraints. To increase efficiency, specialized programs are employed as the standard method of solving these problems. However, these programs are time-consuming and tedious to develop, and hard to maintain and extend [VanHentenryck89].

A common search algorithm in logic languages is *generate and test* which basically means to generate a solution (e.g. a point in the problem space) and test if it is in the set of goal states [Rich83]. Generate and test programs are easy to code, and can be augmented with control information to increase their efficiency to that of a standard backtracking search method [Kowalski79]. However, standard backtracking is still inefficient, showing drastically decreased performance as the problem size grows [VanHentenryck89]. A

refinement using *intelligent backtracking* [Bruynooghe84] attempts to remedy the drawbacks of chronological backtracking by analyzing the source of failures to avoid unproductive choice points. However, there are theoretical limitations implied by using heuristics to find backtracking points [Wolfram86] and additional program overhead. Constraints are only used passively to reduce the search space after failure is detected (*a posteriori*). They are not used to actively avoid failures [Naish85]. In contrast, the other search options, forward checking^A and lookahead^A, prune the search space before failure detection (*a priori*). Such *consistency techniques*, based on *a priori* pruning, are integrated into logic programming to introduce a knowledge representation formalism known as constraint logic programming.

3.2.3 Constraint Logic Programming.

- Constraint logic programming (CLP) languages are based on consistency techniques that originated from Waltz's filtering algorithm [Waltz72]. The concept is to spend more time at each node of the search tree evaluating and removing improbable combinations which will alleviate thrashing [Freuder78, Mackworth77]. Thus, the solution space is reduced before any failure is discovered. Existing CLPs include CHIP [VanHentenryck89] CLP(R) [Jaffar87] and Prolog III [Colmerauer90]. The intention of the work in CLP is to solve, within the logic programming paradigm and in an efficient manner, a class of problems known as constraint satisfaction problems (CSPs).

A CSP is formally defined as follows: Assume there exists a finite set I of variables $\{X_1, X_2, \dots, X_n\}$ which take their values from finite domains D_1, D_2, \dots, D_n and a set of constraints⁷. A constraint $c(X_{i1}, X_{i2}, \dots, X_{ik})$ between k variables from I is a subset of the Cartesian product $D_{i1} \times D_{i2} \times \dots \times D_{ik}$ which specifies a compatible set of values. In other words, a constraint defines a relationship between a set of variables in the problem. CSPs include logic puzzles, edge labelling, and optimization problems. An example of a typical logic puzzle is the SEND MORE MONEY problem (see Rich83 pages 94-99). In this problem, digits from 0 to 9 must be assigned to each letter in such a way that the numerical value for SEND plus the value for MORE is equal to value for MONEY.

⁷The terms *constraint* and *constraint propagation* are used widely to refer to different kinds of techniques for CSPs, unrelated to logic programming, that are used similarly to avoid combinatorial explosion problems. Amongst examples from qualitative reasoning [Kuipers86], planning [Wilkins84] and robotics is a well-known example from interactive computer graphics- SKETCHPAD [Sutherland63]. Sketchpad uses a kind of constraint propagation called value inference which labelled variables with constant values, and then used constraints to find the values of uninstantiated variables from the instantiated variables.

3.3 THE ECHIDNA CLP LANGUAGE.

Constraint logic programming (CLP) has promising potential for application in expert systems technology. ECHIDNA is a new type of constraint logic programming system for model-based expert system applications. Representative of the next generation of expert system shells, it is a synthesis of three technologies: schema knowledge representation, constraint logic programming, and intelligent backtracking via justification-type reason maintenance. The objectives of the next generation expert system shells are: 1) to incorporate richer structured knowledge representation than the flat, unstructured knowledge bases of the mature rule-based expert system technology [Harmon88]; and 2) provide more efficient constraint propagation and intelligent backtracking control structures.

Traditional expert system tools are rule-based. Though this technology is widely used in many different areas, two major deficiencies have been identified. The first problem is their inadequacy in representing knowledge for complex tasks. The second is their procedural inadequacy in applying knowledge [Havens92]. Approaches in rule-based systems to resolve these deficiencies include focus mechanisms, priority rule execution, blackboard architectures, frames [Minsky75], and schemata [Rummelhart76]. However, these approaches move away from the declarative adequacy intended by rule-based programming [Havens92] and introduce knowledge formalisms that are not semantically strong [Hayes81].

In ECHIDNA, the object-oriented schema provides greater descriptive adequacy through composition and specialization hierarchies that are isomorphic to the structure of the domain of the task at hand. Thus, the implicit structure of the knowledge can be used to improve the efficiency of the search process. The underlying logic programming language establishes a strong semantic foundation. Constraints are applied *a priori* to reduce the search space, and solutions to constraints are found by efficient programming techniques using ECHIDNA primitives `INDOMAIN`, `DELETEFF` and `DELETEFFC` (see Sidebottom92b), which are based on those found in `CHIP` [VanHentenryck89]. These primitives assign values to the most constrained variables first. This reduces the amount of the backtracking in the system with earlier failure detection. In an overly constrained situation, the satisfaction of all the constraints will reveal a solution or failure with little or no backtracking. The intelligent backtracking algorithm in ECHIDNA improves on traditional backtracking by reusing the portions of the derivation tree that are unchanged by

the retraction [Havens91]. For more details on these features of ECHIDNA, the reader is referred to [Havens92, Sidebottom92].

The features of ECHIDNA provide a powerful formalism for representation and manipulation of knowledge. The schema structure can potentially represent a wider range of problem domains in a relatively easier and more natural manner. That is, the object-oriented structure of the knowledge bases is better equipped to reflect the real world view of the problem domain, using composition graphs and inheritance hierarchies (see Sidebottom92b page 19). In particular, the interest is in modelling and solving intelligent behaviour problems identified in the introductory paragraph of this chapter.

3.4 CLASSICAL AI PLANNERS.

Intelligent behaviour problems are characterized by general strategies that reduce the computational complexity inherent in the problem. That is, the solution to the task is found by utilizing heuristic knowledge of the problem domain. This thesis explores the manner in which an object-oriented constraint logic programming formalism, specifically that found in ECHIDNA, may be used for simple planning and reactive behaviour of intelligent entities. In particular, the planning process for a sailing action is explored.

The classic AI planner is based on the assumptions made by early planners such as STRIPS and NOAH from the late sixties. The classical definition of the planning problem assumes a state-based representation of the world (i.e. a "snapshot" of the world at a point in time).⁸ The inputs to the planner consist of the world state, actions and problems that need to be solved. The output from the planner is a sequence of *primitive* actions (i.e. actions for which the planner does not know details but are understood by the agents performing the actions in the world environment).

The world state input is generally described by a set of sentences, axioms and constraints. The actions permitted must provide the mapping between the current state of the world and the state of the world after occurrence of the action. Actions are represented by *operators* and *axioms*. Axioms are rules that describe the components of the world that are changed by an operator, or not changed by an operator as in the case of *frame axioms*. Axioms in the classical BLOCKS world domain describe the state of the world after operations such as stacking and unstacking. The BLOCKS world consists of a set of

⁸Alternative representations include event-based representations [Lansky88].

blocks that may be placed on top of each other by a robot arm. A BLOCKS world planner must plan the operations needed to move the blocks from the initial world state to the goal state. The following is an example of an axiom from a planner for the BLOCKS world:

$$\text{CLEAR}(x,s) \text{ and } \text{ON}(x,y,s) \rightarrow \text{HOLDING}(x, \text{DO}(\text{UNSTACK}(x,y), s)) \text{ and } \text{CLEAR}(y, \text{DO}(\text{UNSTACK}(x,y), s)).$$

The axiom given describes the changes in the world state after an UNSTACK operation. It specifies that unstacking block x from block y will clear block y and result in block x being held by the robot arm. The conditions for this action are that block x must be clear and block x must initially be on top of block y.

The problems to be solved by the planner are the *goals* of the planning process. The classical planning process then requires that the sequence of actions produced as output by the planner, when applied to the initial world state, will achieve the specified goals. Planning, therefore, is viewed in the classical sense as *a search through the space of operator applications and plan orderings* [Wilkins88]. Desired features of a classical AI planner are given in the table in Figure 3.1.

<u>Desired Features of a Classical AI Planner from [Wilkins88]:</u>	
• Causal Theory:	A causal theory of a domain is the set of axioms that describe the causal connections in a domain.
• Nonlinearity:	The ability of a planner to handle actions that are unordered with respect to each other and may possibly be parallel actions. This characteristic is important for efficiency (i.e. to avoid searching all possible plan orderings).
• Plan Hierarchy:	The ability to reason at different levels of abstraction, both in the planning process and in the description of the domain. This avoids searching all possible detailed plans.
• Plan Variables:	Planning variables allow certain entities in a plan to be left unspecified during the planning process to avoid searching every plan instantiation.
• Constraints:	Use of constraints to reduce the search through every possible plan instance.
• Replanning:	The ability to reuse or modify the current plan without having to replan from the start after an unexpected event which is typical of the real world.
• Domain Independence:	Being domain independent allows a planner to be readily adapted to new problems. ⁹ However, domain knowledge is a necessity. Thus, domain independent planners provide a mechanism (i.e. a knowledge representation language) to encode domain-specific knowledge.

Figure 3.1 Desired Characteristics of AI Planners

⁹Critics of domain independent planners claim that such an approach is weak and requires a considerable amount of domain knowledge to do planning. Critics of domain specific systems claim that such "planning systems" avoid the real problem related to reasoning about actions in a general manner.

The causal theory is important for deducing the context-dependent effects necessary to solve the frame problem. One aspect of the frame problem is representing all the facts that do not change after the action (e.g. tacking a sailboat does not change the colour of the sky). Also, the facts that do change are context-dependent (i.e. the effects of an action are dependent on the situation in which the actions are performed). Nonlinear planners have the additional difficulty of recognizing and resolving conflicts between actions (i.e. interferences of one action upon another action) in the quest for correct plan generation. Other approaches to reasoning about actions use *unrestricted logics* (e.g. circumscription [McCarthy80]). Though unrestricted logics are more expressive and can withstand more rigorous analysis, they suffer from inherent computational difficulties and unclear usage.

Some of the early planners, as identified in [Rich83] and [Wilkins88], include STRIPS, HACKER, ABSTRIPS, NOAH, NONLIN, DEVISER, MOLGEN and SIPES. The (strict) STRIPS assumption, is the basis of all subsequent solutions to the frame problem in classical planners. The (strict) STRIPS assumption is that no predicate will change due to an event unless the operator for the event *explicitly* states that the value of the predicate changed. This assumption is the heart of the achieved efficiency, and limitation, of classical planners. ABSTRIPS introduced hierarchical planning (different abstraction levels in the planning process). HACKER was the first to introduce multiple goals. NOAH, the first true classical planner, developed the concept of plan critics for plan modification due to the complexity of nonlinearity and planning variables. NONLIN built on NOAH by adding backtracking and more powerful plan critics. DEVISER incorporated temporal reasoning capabilities into NONLIN. MOLGEN, though domain-specific, successfully introduced the use of constraints and metaplanning to help control searching. SIPES, developed in the mid eighties, is the first classical planner to use causal theory, permit general constraints and do interesting replanning.

Note that the classic AI planner is not *reactive*. The classic planner assumes that the initial world does not change during the *plan time* when the planning process is being constructed. The plan time is different from the *execution time*. A planning system is reactive if it can react in an acceptable amount of time to any changes that occur in the world while the system is running. Proponents of reactive systems, reasoning systems without planning, argue that there is no distinction between planning and execution. That is, planning is not necessary; only reactive controllers are necessary. Classical planners are extremely slow and are not effective in real-time environments. As one puts it, "the classical planner might be run over by a train while it is planning" [Wilkins88]. Additional

evidence of the importance of reactive systems is that humans are often seen as simply reacting to external stimulus without thinking ahead. It is currently believed that in most environments both a planning component and a reactive control system are needed.¹⁰

In this thesis, the planner for the proposed conceptual framework is implemented in the constraint logic programming language of the ECHIDNA reasoning system. It does not strive to achieve the features desired in a classical AI planner (as listed in Figure 3.1). However, it does investigate the relationship between planning and plan execution (reaction component) in a constraint-based implementation and animation environment.

¹⁰Reactivity is necessary for autonomous agents in the world. However, it is also argued that even reactive autonomous agents must do some planning in advance.

4.0 CONCEPTUAL DESIGN.

4.1 OVERVIEW.

An intelligent entity has the ability to reason about its world and act in accordance with the state of its world. "Intelligent entities seem to *anticipate* their environments and the consequences of their actions. They act as if they know, in a sense, what the results would be. We can account for this anticipatory behaviour by assuming that intelligent entities themselves possess knowledge of their environments." [Genesereth87 page 2]

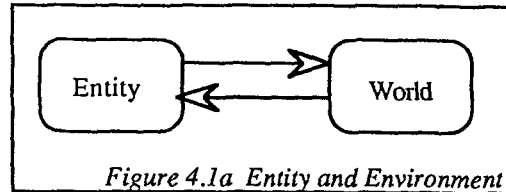


Figure 4.1a Entity and Environment

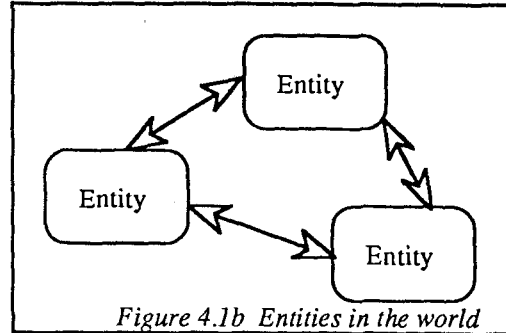


Figure 4.1b Entities in the world

For animation, capturing this anticipatory behaviour of intelligent agents can be quite daunting. The interactions between the entity and its world (see Figure 4.1), including other entities, grow quickly in complexity. In an environment with multiple entities, each entity's world is slightly different, and thus, each entity's behaviour differs. Animating the behaviour of a large number of such entities by traditional methods (e.g. keyframing) is very labour intensive. Also, there is little allowance for alterations or experimentation. To facilitate the animation task, alternative high-level approaches are needed for behavioural animation. This chapter proposes a conceptual framework for an integrated reasoning and animation system for situations with multiple intelligent entities.

Group animation is automated by defining the behavioural profile of the individual entities, the tasks that they are capable of performing, and the layout of the environment in which the entities interact. The behavioural profile describes how an object relates to its environment (i.e. other physical objects, both moving and static, and changes in environmental factors). The task profile describes the actions that the entity is capable of performing, and the preconditions^A necessary before an action can be undertaken. The resulting observable motion is then a reflection of the object's behaviour in response to internal and external stimuli. Different animation scenarios with multiple entities can be created by changing the behavioural and task profiles of all or some of the objects, rearranging the layout of the environment and varying the values of environmental factors.

The directive of this thesis work is to model the reasoning process of the intelligent entities, or *agents*, controlling the physical object under motion in a specific domain. The proposed system does not aim to model realistic motion at the physical level. There is a clear distinction between the physical description of motion and the behavioural description. The physical description is responsible for providing procedures, or *motion units*, for desired movement which may strive to be realistic or impressionistic. The motion units may be created by keyframing, kinematics, dynamics or statistical simulation techniques. Motion description at this level refers to manipulation of positions coordinates, joint angles, forces and torques. These variables pertain to the low-level mechanics of motion (i.e. the *how* aspect of an action).

Whereas, the behavioural description of motion is responsible for determining *when* particular motion units should be attempted. The knowledge bases encode the criteria and consequences of enacting a particular action (i.e. the *why* aspect of motion). Thus, when given a goal (i.e. *what* should be attempted), the agents will deduce a feasible plan of actions since they understand the requirements and effects of each action. During the animation of the planned sequence of actions, the agents should also determine when to do plan modification or replanning to ensure the given goal is achieved.

An animation of behavioural motion is characterized by the following properties: 1) internal motivation and knowledge of the intelligent object is reflected in its course of action; 2) interaction amongst the objects themselves; 3) interplay between environmental factors and the object; 4) multiple objects with similar behavioural rules; 5) a dynamic environment; and 6) a degree of nondeterminism (i.e. there are many possible outcomes for a given layout, a number of objects and their behaviour). Examples include scenarios such as pedestrians in a street scene, rush hour traffic at a controlled intersection, sailboats in a race course, etc.

An ideal framework for such an integrated reasoning and animation system should provide strong knowledge representation and reasoning capabilities in a modular, expandable and reusable framework. It should also provide facilities for high-level, interactive and intuitive control of behavioural motion for individuals and groups. The overall framework consists of two main components: the reasoning component and the display component. Each component consists of separate modules. The main modules (see Figure 4.2) are: 1) the reasoning system and knowledge bases; 2) the object and environment modelling module; 3) the motion description module; 4) output modules for

display; and 5) the interface module which provides the linkage between the separate modules and particularly between the reasoning and display modules.

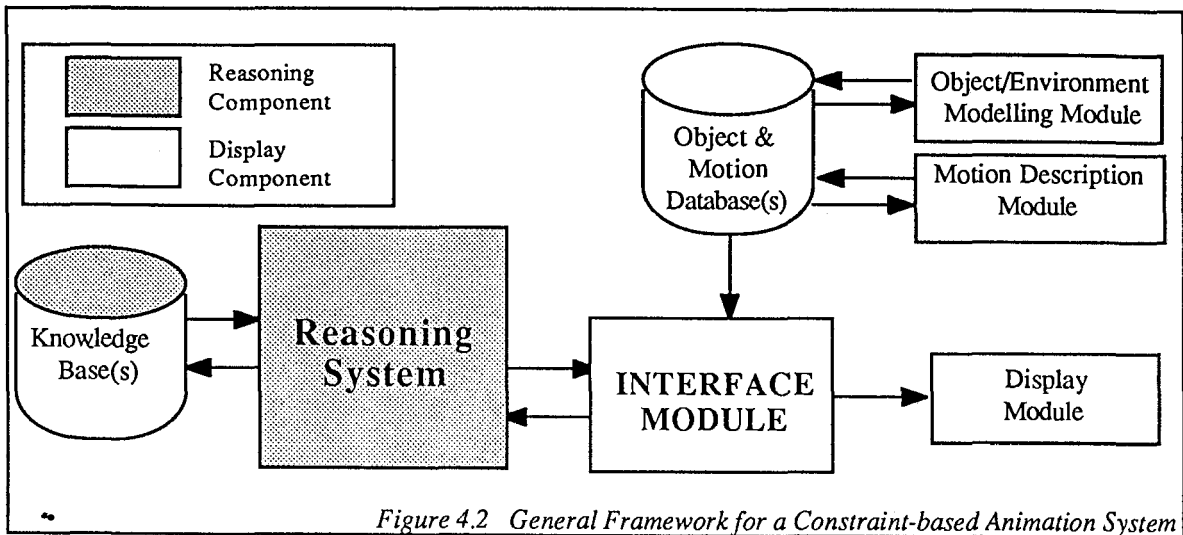


Figure 4.2 General Framework for a Constraint-based Animation System

The reasoning system chosen is the ECHIDNA model-based constraint reasoning system. The knowledge of the agents and formal relationships in the world are encoded with the ECHIDNA CLP language. The proposed framework provides an object-oriented structure to handle the complexity of diverse knowledge bases. Furthermore, the proposed structure attempts to apply various guidelines that may lead to greater efficiency and speed in constraint processing. An objective of this initial development is to investigate how to best employ the features of the ECHIDNA CLP language and expert systems shell in building knowledge bases for animation.

4.2 WORLD REPRESENTATION.

The world must be represented in both components (i.e. the reasoning and animation components) of the constraint-based animation framework. The world consists of static and moving objects, some of which are intelligent. Each intelligent object has both a reasoning part and a physical part. The physical part, used by the display component, is the physical description of the object (e.g. the polygonal model of a human body) and the mechanics of its motion units (e.g. a sequence of keyframes for a movement such as jumping). The reasoning part, represented in the reasoning component, controls the application of the motion units. Both parts of an object contribute to the determination of the object's motion. The reasoning component part decides which action and how it should be performed. The display component part enact the underlying details in the motion unit.

4.2.1 Objects.

All objects in the world belong to the generic schema class `PHYSICAL_OBJECT`¹¹ (see Figure 4.3) which is then divided into `MOVING_OBJECTS` and `STATIC_OBJECTS`. Each physical object has a position, type and state. The position is its current calculated location in the world. State variables are found through the reasoning process and used to calculate changes in position. The state represents an object's internal configuration which is dependent on its type. For example, a moving object has a direction and velocity. States can be further refined in terms of its specific object type (i.e. the particular kind of moving object). For example, a sailboat object would have a state variable for its sail setting; a car object would not.

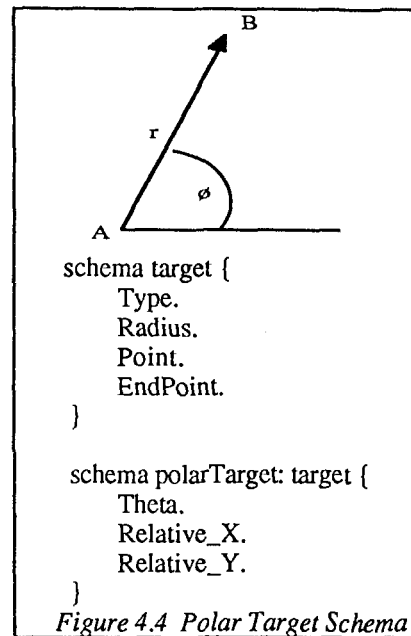
```

schema physical_object {
    Position.
    Object Type.
    State.
}

```

Figure 4.3 Physical Object Schema

The hierarchical organization of objects for inheritance and polymorphism^A facilitates the handling of unidentified objects. Each object has default methods for interacting with generic static and moving obstacles. This is consistent with the real world where intelligent entities are capable of generalizations of objects. An unidentified object in this context would be one for which there is no specific handling information. Thus, when an unidentified object approaches, generalizations are made about its size and speed. An appropriate action is then taken (e.g. get out the way if it is a large and fast).



4.2.2 Targets.

The `TARGET` schema identifies a point in world space for a desired action. The generic target schemata are point targets, area targets¹² and polar targets (see Figure 4.4). Targets in general relate the heading of a moving object in world space to a specific point or area. An area target defines a circular area with a specified radius. The targets are currently defined as two-dimensional representations.

¹¹For the purpose of identification, names of schemata and variables in the ECHIDNA knowledge bases are capitalized in the text of this thesis. Examples given throughout this document are based on the ECHIDNA CLP language syntax. Readers unfamiliar with ECHIDNA should refer to [Sidebottom92b] for details.

¹²Area targets do work when implemented with interval variables. Point targets using intervals are not truly point targets unless the precision is set high enough. Refer to chapter eight for details.

A polar target (Figure 4.4) relates two world points (i.e. point A and end point B) to each other using angle \emptyset (i.e. the THETA variable in the POLARTARGET schema in Figure 4.4) and/or radius r . This representation is useful for defining heading and distance relationships between two objects. Introducing the notion of a polar target simplifies the coding of relationships between moving objects involving distance, heading and positions.

4.2.3 Motion Units.

The motion modelling module defines the physical motion of the objects. An object may have a number of different *motion units* which correspond to actions in the reasoning component. For example, a car may have four motion units: drive, turn, start and stop. The motion units can be created by keyframing or procedural methods with kinematics, dynamics or statistical models. They are used by the application agents to compute successive positions of the object in world space over time. In this manner, computationally intensive calculations are done outside of the ECHIDNA reasoning system with the most appropriate numerical techniques.

The separation of the position computations enforce a higher view of motion. The objective is to reason about motion in terms of its descriptive parameters [Morawetz90] such as its path, rotations, goals or dynamics [Badler86]. The reasoning component only needs to know the new position and whether the motion is interruptible. This makes it easier to replace motion units with improved models at a later date. The exact details of how the motion units are implemented depend on the specific animation domain.

4.3 THE REASONING COMPONENT.

-- The reasoning component, consisting of the ECHIDNA reasoning engine and its CLP knowledge bases, provides powerful knowledge representation by incorporating inheritance, overloading and modularity in its knowledge bases [Havens92]. The proposed knowledge structure takes advantage of this greater expressive power to provide an environment that is adaptable to different application domains, and makes it simple to add new objects to an existing application.

4.3.1 Knowledge Representation Structure.

As well as modularity and ease of expansion, the proposed knowledge base structure strives for efficiency by using *interval* variables (see Sidebottom92b) and by reducing the

resources consumed by the constraints. One measure of consumption is the cardinality of the constraint. The lower the cardinality, the lesser the amount of resources used. Simpler constraints, such as those with fewer variables and smaller domains, are believed to contribute to faster resolution time because each constraint successively reduces the domain space of variables involved for the next constraint.

Thus, the knowledge base is organized in small knowledge units called *morsels*. Each morsel relates variables and objects to each other, and presents a format to organize complexity. Morsels are implemented as schema classes, allowing for specialization and inheritance. The morsels are used by *knowledge agents*, the agents associated with an intelligent object. Varying the combinations of morsels for an agent results in different behavioural motion. In other words, what the agent knows is reflected in its motion.

Knowledge Units.

The behaviour of objects is represented in knowledge bases as morsels. Morsels have associated *morsel variables* which are variables that can be related to other variables. Each morsel operates on one or more morsel variables which are bound to corresponding object variables. In Figure 4.5, OBJECT_VARIABLE_A is bound to MORSEL_VARIABLE_A so that the morsel constraints can be applied to the object variable. Each object variable may be bound to multiple morsel variables. Thus, the relationships between object variables are stated

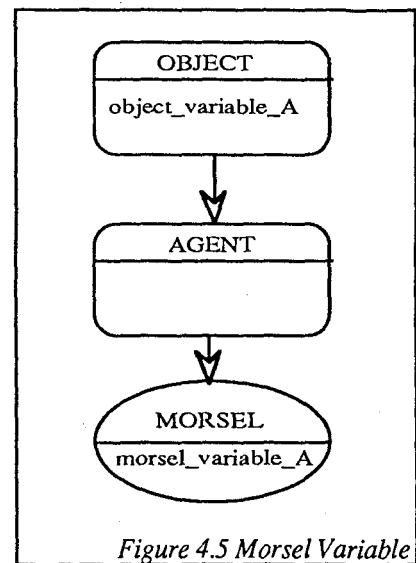


Figure 4.5 Morsel Variable

in multiple smaller constraint expressions defined in a set of morsels. This reduces the cardinality of the constraints and improves the efficiency of constraint propagation. New relationships can be added by introducing new morsels to the ECHIDNA knowledge bases.

There are three types of methods associated with the activation of constraints in knowledge morsels: BIND, APPLY and ASSIGN. The BIND method binds an object variable to a morsel variable of a specified morsel. The APPLY method applies the constraints defined in the bound morsel to the object variable. This propagates constraints to reduce the domains of related object variables. The ASSIGN method assigns values to variables using INDOMAIN and SPLIT primitives for discrete and interval variables respectively. Backtracking occurs until a consistent state is found for all object variables.

```

schema testmorsel: morsel
{
  int Morsel_Variable.

  morsel(int Object_Var) :-
    Morsel_Variable = Object_Var.

  order apply.
  apply:- Morsel_Variable =:= 10.
  apply:- Morsel_Variable < 20.
  apply.

  assign:- indomain(Morsel_Variable).
}

```

Figure 4.6 Example Morsel

Morsels can be used to encode priorities and heuristics as shown in the *testmorsel* schema (see Figure 4.6). This provides a mechanism in which morsels can be somewhat prioritized using the ORDER function in ECHIDNA. A morsel that has high priority will have only one APPLY method, and thus gives no clause for backtracking. If there is an inconsistency, another morsel operating on the same object variable with multiple APPLY clauses will have to backtrack to satisfy all constraints on the object variable.

The final APPLY definition in the *testmorsel* schema indicates that the constraints in the morsel need not be applied at all and no failure will be outstanding. This represents a form of heuristic knowledge. Heuristics are "rules of thumb" that are *usually* true [Sidebottom92a]. In *testmorsel*, the heuristic states that MORSEL_VARIABLE should be unified with a value of 10. If this leads to an inconsistent state, then assign any value less than 20. Otherwise do not apply any constraint from *testmorsel* to the variable.

The primitive *morsel* schema is the basis for three other categories of morsels: object morsels, plan morsels and reaction morsels. Object morsels define relationships associated with the internal operation of the object and with environmental factors that determine its internal state. For example, a sailboat will have an object morsel relating the position of the sail to the direction of the wind. Plan morsels define the relationships between goals and actions, and encode the strategic knowledge for planning. These morsels are used in the planning process to find an appropriate set of actions to accomplish a goal (e.g. reach a specified destination). Reaction morsels for plan execution define the relationships between the object and other static or moving objects in the domain. Reaction morsels include "right of way" constraints to avoid collision with other moving obstacles.

Knowledge morsels can be selectively added to an object. Varying the morsel composition amongst objects will lead to different behavioural and task profiles. For example, in animation of car traffic, an automobile object may know the rules for traffic lights (i.e. green means "go" and red means "stop") but may have no strategic knowledge for reaching a particular destination. Such a scenario would include a car with a driver who has no idea of where to travel, and just drives around aimlessly obeying traffic lights.

Knowledge Agents.

Knowledge morsels are used by *knowledge agents*. Agents^A represent the intelligent entities controlling the physical object under motion. As shown in Figure 4.7a, there are three basic types of knowledge agents used in NSAIL: planning, implementation and perception. The three agents control the behavioural motion of an object by responding to stimuli from their environment. Overall object behaviour is thus influenced by the collective knowledge of these agents. It is not necessary to have all three agents associated with every intelligent object in the animation environment.

Each agent has two sides: a side that resides in the reasoning component; and a side that resides in the display component (see Figure 4.7b). An agent may have greater functionality on one side than on the other. For example, the planning process performed by the planning agent occurs on the reasoning component side. The display component side of the planning agent only needs to initiate the planning process with the top-level goal specified by the user.

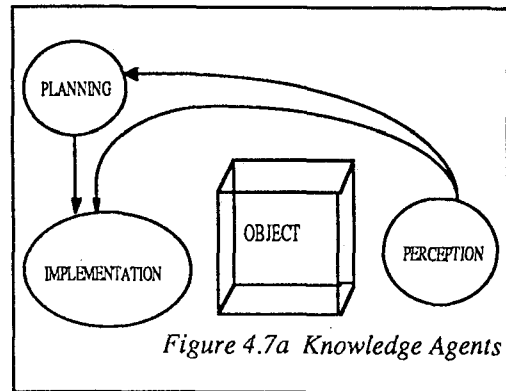


Figure 4.7a Knowledge Agents

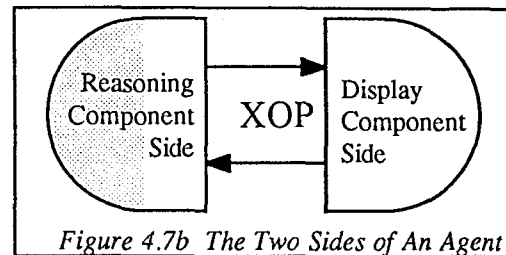


Figure 4.7b The Two Sides of An Agent

4.3.2 The Planning Agent.

The planning agent is responsible for development of a *plan*, a set of actions or motion units needed to attain the top-level goal. The implementation agent is responsible for executing the plan which constrains its reasoning process as it reacts to events from its dynamic environment. It detects goal satisfaction and prepares for the execution of the next step in the plan. The perception agent monitors the external environment by maintaining a *panic box*, which is a list of hazardous objects in the immediate vicinity (e.g. those that may be on a collision course with the agent's object). The perception agent sends the contents of the panic box to the planning agent as input into the planner.

A plan goal consists of a destination, an action to be performed at the given destination, and a post action state (see Figure 4.8). It is possible to specify only an action or destination as a plan goal. Note that this is a task planner problem in which the tasks to be performed dictate the path of the object while achieving the given goal.

The high-level goal is assigned to the object by the display component side of the planning agent. The generated plan from the backward planner^A used by the planning agent is a list of *plan nodes*. The final plan node in the list is the given goal node (i.e. plan node P0 in Figure 4.8). The first plan node represents the initial state at the start of the planning process. The remaining nodes are the *plan steps* required for attaining the state of the final goal node from the initial state node. The planning algorithm views each node as a set of constraints on the next

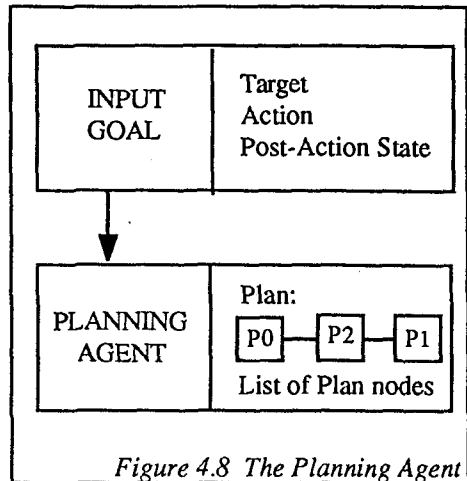


Figure 4.8 The Planning Agent

node in the list. The basic relationship between two consecutive plan nodes is that they are not exactly the same in order to avoid infinite looping during planning¹³. The remaining constraints relating plan nodes are from the domain-dependent object and plan morsels.

The skeleton of the planning algorithm is shown in Figure 4.9. It recursively applies constraints to the plan nodes in the plan list, and assigns values to them until a solution is found.¹⁴ Constraints are defined in the methods contained within morsel schemata. A method may have multiple clauses, indicating the presence of a choice point in the solution space. For example, the PLANSTEP method has two clauses, establishing a choice point with two choices of which only one may be active at any one time. The ORDER primitive (see Sidebottom92b) ensures that the backtracking order of the clauses for the PLANSTEP method (see Figure 4.9) corresponds to the order given in the knowledge base. A clause when found to be inconsistent is marked as "no good" at its choice point and the active choice is set to a following clause in the knowledge base. If elaboration elsewhere causes the "no good" marker to be removed, the active choice will automatically be reset to that clause. In general, the active choice is reset to its closest valid choice (i.e. defined earlier in the knowledge base without a "no good" marker). In this manner, the ORDER primitive controls when a new plan node is added to the plan list.

PLANSTEP clause B adds another node to the plan list and applies constraints to relate the new node (identified as plan node P_c) to its two neighboring nodes in the list. A

¹³This is a very simple planner. For more complex planning problems, additional expertise may be needed to recognize inefficient loops or abnormal situations. This can be done by *critics* as in classical AI planners such as NOAH and SIPE. Critics are domain-dependent and would be implemented as morsels. The difficulty would be in determining when to apply these morsels.

¹⁴Note all constraints must be applied before any values are assigned. The planning algorithm accounts for this but thrashing can still occur during implementation of the plan. See chapter eight.

new node is only added when there is no plan to be found with the current number of nodes in the plan list. This means that another action is needed in the plan to accomplish the given goal. When values are assigned to the plan nodes in the plan list which has the newly added node, backtracking may occur to relate different constraints to the previously-created nodes.

The hierarchy of plan nodes given in Figure 4.10 shows the relationship between nodes in the plan list. Each level in the tree depicts the current contents of the plan list after a plan step where an additional node is added to the plan list. For example, in Figure 4.10, the root of the tree consists of the final and initial nodes only. The second level of the tree shows that a newly created node P2 is related both to node P0 and node P1. Node P2 is the precondition node for the final goal node P0, and is at the same time, the goal node for initial state node P1. The third level of the tree shows the relationships when node P3 is added. The actual plan list with four nodes is shown on the right-hand side of Figure 4.10.

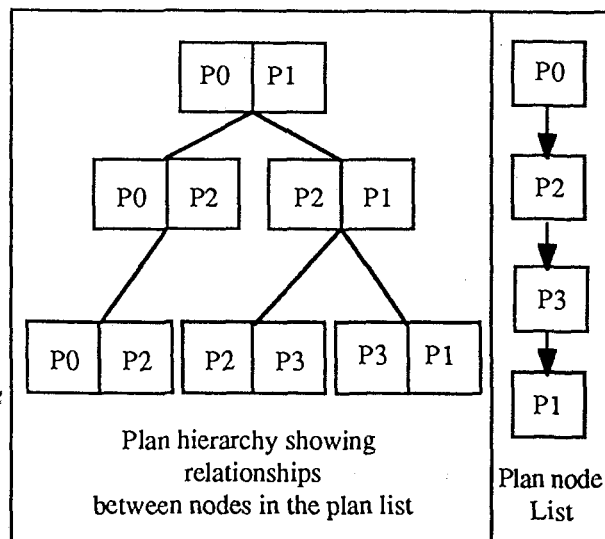
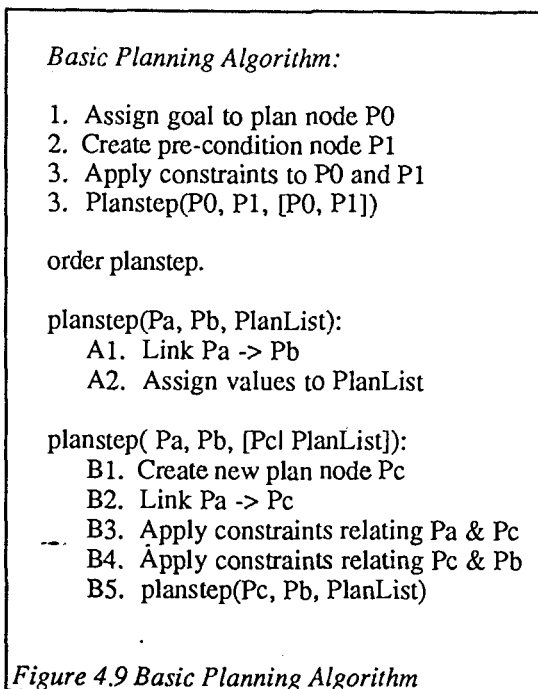
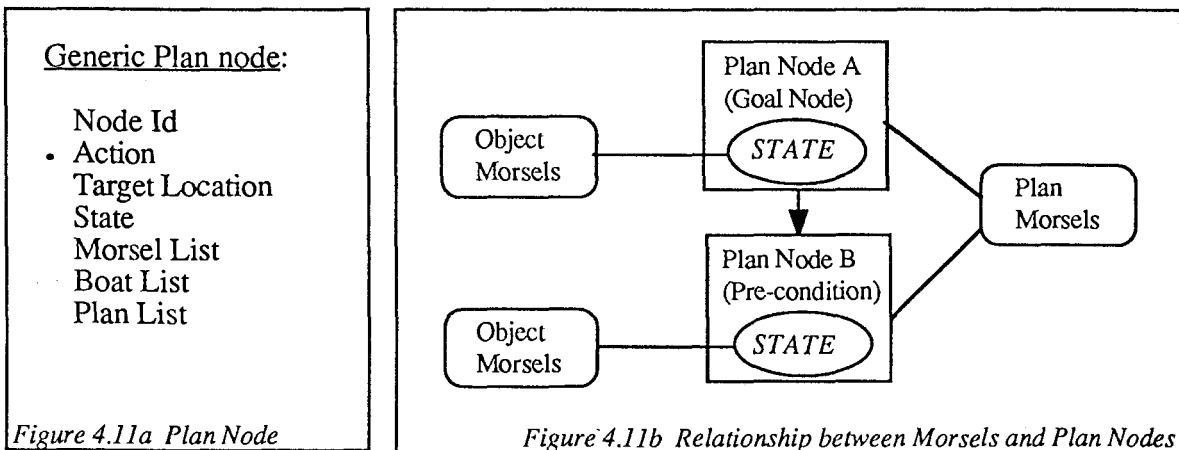


Figure 4.10 Plan List Hierarchy

Note: Since this is a backward planner, execution starts with plan node P1 and goes back towards plan node P0 in the list shown above.

Applying constraints in the planning algorithm means binding morsels to the plan nodes. The morsels, identified in the MORSEL_LIST of each plan node, are assigned during creation of the physical object. Object morsels constrain the STATE schema instance in the plan node. The STATE schema instance of each plan node is instantiated with the specialized STATE schema definition in the associated object. Plan morsels for the object are bound to the goal plan node (plan node A in Figure 4.11), but relate the two plan nodes to each other. Therefore, plan nodes in the list will be related to each other. A plan then is an assignment of actions, targets and states that establishes consistency amongst active constraints on all plan nodes.



The effectiveness of the planner, as in most classical AI planners, lies in the development of the planning knowledge. With the proposed constraint-based planner, consideration must also be given to the order of constraint application and value assignment. One guideline in constraint satisfaction problems with ECHIDNA is to apply *proper constraints* first to reduce the number of backtracking choice points¹⁵. Proper constraints refer to methods with only a single clause, and are defined with other single clause methods. Hence there are no choice points introduced. Value assignment should start with the most constrained variables (using something similar to LABELLEFF primitive in the CHIP language [vanHentenryck89]). Assignment of values to discrete variables should be performed before assignment of values to interval variables. Splitting an interval variable adds choice points to the solution space¹⁶. Thus, the chosen order of assignment for plan variables is: action, state and target. Within the STATE variable, assignment begins with discrete variables as well.

¹⁵This was discovered at a rather late stage in this thesis work.

¹⁶The splits are binary, and the number of splits is the same as the precision. If the precision is 30, then each variable does 30 splits. Thus, pruning with constraints is important at higher precision values.

Plan modification and replanning are two desirable features of planners [Wilkins88]. In the proposed framework, these two properties are reflected in the degree of backtracking that occurs when a plan step fails during execution of the current plan. Since the initial planning is based on the starting position of the object, replanning produces a new plan using the current position of the object. The implementation agent signals the planning agent when a plan step fails at the top level.

4.3.3 The Implementation Agent.

The implementation agent handles execution of the generated plan. The implementation agent is equally active in both components of the system: the ECHIDNA reasoning component and the application display component. The resulting animation is the product of the execution of plans by implementation agents. Execution of the plan alternates between reasoning and display processes. The implementation agent must first find a consistent state using its reasoning side, and then calculate and output its new position using its display side. On the reasoning side, the agent maintains a state that is consistent with constraints from the plan, object morsels and reaction morsels (see Figure 4.12). The reaction morsels deal with the collision avoidance relationships and the dynamic environment in which the objects are being animated.

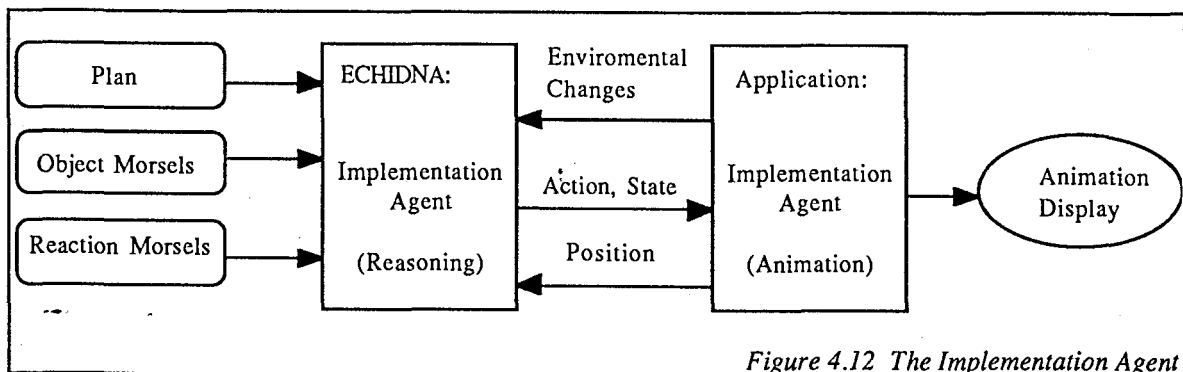


Figure 4.12 The Implementation Agent

The reasoning and display processes are repeated at each *time step* of the animation. That is, at each time step, the implementation agent is responsible for relating the new position to constraints defined the associated morsels. The actual time between time steps, represented by time units or *ticks* of 1/30 of a second, is set by the user for real-time display or batch playback (i.e. calculate the changes for a particular period of time and play back after all the calculations are done). Note that real-time display is not yet feasible due to the high computational time required by the reasoning tasks in the NSAIL system.

The basic implementation algorithm is listed in Figure 4.13. The animation process is initiated from the display component side since ECHIDNA goals can only be undone at the top level of the ECHIDNA interface. Once the external object protocol (XOP) connection is established between the two sides of the implementation agent, changes in linked ECHIDNA variables associated with the object state are automatically propagated to the display component side. The XOP allows *external methods* to be activated from the ECHIDNA reasoning system. External methods are methods defined outside of ECHIDNA, and encoded in the programming language of the display component (e.g. a procedural language such as C++ or C). When an external method is invoked, a persistent data link is created for the arguments of the method.

Basic Animation Algorithm:

Initialization for each intelligent object:

- I1. Apply plan node constraints
- I2. Apply object morsels
- I3. Apply reaction morsels
- I4. Establish XOP connection

Loop for each time step:

- L1. Undo previous object assignments
- L2. Undo previous position goal
- L3. For each object, assign values
- L4. Calculate change in position (app)
- L5. Update animation display
- L6. Bind position on ECHIDNA side

Figure 4.13 Basic Animation Algorithm

The implementation agent recognizes when the plan goal is attained. This implies that the object reached the plan target point, completed the plan action and is consistent with the post-action plan state. When a goal plan node P is bound to an object (initialization step I1 in Figure 4.13), the object is in a configuration that is consistent with the precondition node (i.e. the plan node prior to node P in the plan list for the object). Achievement of the goal is done in two stages using flags set by the display component side to indicate the completion of each stage.

Goal Achievement Morsel

```

{ 1 .. 4 } Goal_Flag.

apply :- relate_flag(Goal_Flag).

relate_flag(1) :- do_target constraints
relate_flag(2) :- do_action constraints
relate_flag(3) :- do_state constraints
relate_flag(4) :- signal goal completion

do_target :-
    if position of object == target
    position, signal application side of
    agent to change flag to 2.

do_action:
    if action completed, signal application
    side to set flag to 3.

do_state:
    Unify object state with post- action
    plan state and signal application to set
    flag to 4.

```

Figure 4.14 Goal Achievement Morsel

The goal achievement morsel is a reaction morsel that relates the object state to the current plan node. To achieve the goal plan, the object must first move towards the goal target. This is target achievement. At the target, a signal (via an external method) is sent to the application side of the agent. At the next time step, the application side will undo the top-level flag goal (along with the position goal) and redo it with the next flag value (see Figure 4.14). This causes the constraints for action completion to be activated. The action completion check is also an external method since the information is relayed by motion units on the application side. This is repeated for the post-action object state and finally, for overall plan goal achievement. At this final point, the implementation agent undoes the top-level plan node goal and retrieves the next plan node from the plan list.

While progressing towards the plan goal, the implementation agent must also manoeuvre around moving obstacles not considered during the planning stage. This is handled by the reaction morsels. The implementation agent only processes the objects in its panic box. The panic box, maintained by a perception agent, contains obstacles that may have an impact on the execution of the plan. For each object in the panic box, an appropriate reaction morsel is instantiated to apply necessary constraints. Thus, constraints dealing with a particular object type are only activated upon perception by the agents.

Using flags set externally from ECHIDNA, requires a mechanism for mutual exclusion. When goals are undone from the display component side, one side effect is backtracking into flag setting clauses of the RELATE_FLAG method. When a flag is set, it cannot be changed to a previous value. Thus, there needs to be a mechanism to ensure that backtracking into such clauses only occurs when the position variables are bound and a valid relationship is defined between the position and target location. Semaphores flags are used to serve this purpose. The agents on the display component side, sets a semaphore flag before undoing the GOAL_FLAG or object POSITION variables.

4.3.4 The Perceptual Agent.

The perception agent is responsible for extracting information from the environment about obstacles that must be handled by the implementation agent. It is only concerned with objects that have possible impact on its associated object (e.g. those on a possible collision course). Collision detection is actually performed by the display component half of the perception agent since path collision computations are easier to program outside of ECHIDNA. The reasoning half of the perception agent serves as the keeper of the panic box for the other agents on the ECHIDNA side.

The perception agent checks the panic box at each time step and communicates any occurrences of change to the other agents. The display component side binds the panic box as a top-level goal to ECHIDNA. Whenever, the panic box changes (i.e. an object leaves or enters the panic box), the panic box goal is redone with the new contents. Thus, at each time step, there may be a series of top-level goals waiting to be redone.

4.4 THE INTERFACE AND DISPLAY COMPONENT.

The application specific interface module on the display component side is closely integrated with the reasoning component. It is responsible for creating the objects and environment layout for the animation, handling communication with the ECHIDNA XOP for the knowledge agents, maintaining the dual nature of the knowledge agents, initializing objects for planning and animation, calculating the position of objects at each time step using pre-defined motion units, and updating the animation display. It may also provide an interface to the animator for interactive control of the animation through manipulation of the environment (e.g. changing environmental factors such as wind velocity and direction). The interface module is the link with other modules for sophisticated rendering and modelling capabilities. Its main responsibility though is to present a single interface to the integrated reasoning and animation system for specifying and controlling behavioural motion of intelligent entities.

5.0 NSAIL IMPLEMENTATION OVERVIEW.

5.1 INTRODUCTION.

The Sailing Scenario.

It's a gusty day on the bay. Skip checks the camber of the mainsail and decides that it should be flatter for the expected wind conditions. As she tightens the battens of the mainsail, Skip looks out across the water... whitecaps at the mouth of the bay. She will have to tack close at the second mark and head high to catch the stronger breeze along the shore of the beach. The first horn sounds as she pushes the boat into the waves. Rudders down. Locked. Jib in. Mainsheets. The hobie picks up speed as it sails out of the wind shadow from the wharf. A sudden gust of wind, and the windward hull lifts. Skip lets the mainsail out and hikes out on the trapeze. Boat on port side. Skip heads upwind and passes behind the oncoming boat.

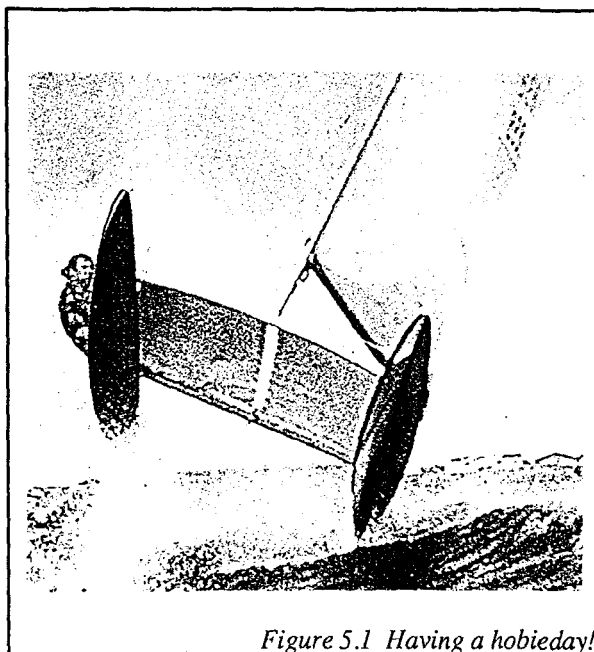


Figure 5.1 Having a hoberday!

5.1.1 Motivation.

NSAIL is an animation interface for a sailing domain. NSAIL is based on the integrated reasoning and animation framework proposed in Chapter four. The objectives of the implementation are: 1) test the proposed constraint-based methodology for behavioural motion control; 2) gain further insight into the use of the ECHIDNA expert system for computer animation; and 3) create an animation of an interesting, nontrivial environment.

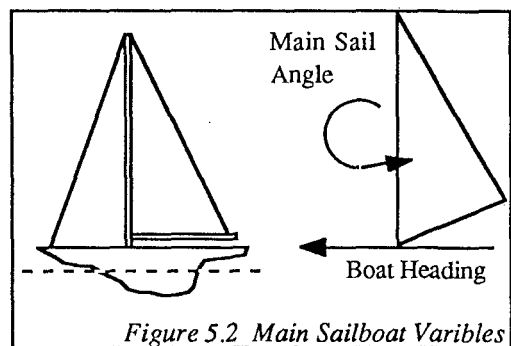
A sailing animation is well-suited for the proposed constraint-based framework since there is a natural hierarchical view of motion control for a sailboat, and the interaction between the sailboat and dynamic environment is clearly visible. There are established navigational objects (e.g. race markers, channel markers, warning buoys, etc.) which dictate definite relationships between themselves and the boat heading. There are well defined, if not widely obeyed, rules for determining the right of way when boats approach each other. The division of responsibilities on a sailboat can be easily represented with the three knowledge agents for planning, implementation and perception. It should be noted, however, that the sailing model for NSAIL is greatly simplified in order to demonstrate the concepts proposed, and does not implement the vast heuristic knowledge in the form of personal theories regarding sailboat performance and sailing strategy!

At the physical level, the motion of a sailboat can be expressed as the result of complex opposing systems of aerodynamic and hydrodynamic forces. However, the motion of sailboats as seen by spectators on the beach is not wholly and solely dependent on the interaction of wind and sail. At a higher level, the course of the sailboat is driven by the intelligent entities at the helm and winch, who are constantly adjusting the boat heading and sail shape in response to continually changing environmental conditions. Thus, the boat, the environment, and the expertise and intentions of the crew impact the observable motion of the sailboat.

5.2 THE SAILING MODEL.

5.2.1 The Sailboat.

The NSAIL boat, at the lowest level of abstraction, is basically a boat heading (see Figure 5.2). All reasoning and manipulations eventually lead to an adjustment of the boat heading. All other NSAIL boat variables are dependent on the boat heading and the wind direction.



The sail angle plays a secondary role. It is related to the wind and boat heading, but can also be manipulated independently. For example, the boat heading relative to the wind dictates the side of the boat where the sail should be, but the exact position can be set by the implementation agent. In NSAIL, the sail angle is used to regulate the speed of the boat, and to infuse a bit of variation for boat performance. If the sail angle is not optimally set, the boat sails at less than optimal speed. In some cases, an inappropriate sail setting leads to a boat capsize.

BOAT SYMBOLS	
θ	Boat Heading
β	True Course
μ	Sail Angle
∂	Wind Direction

Figure 5.3a

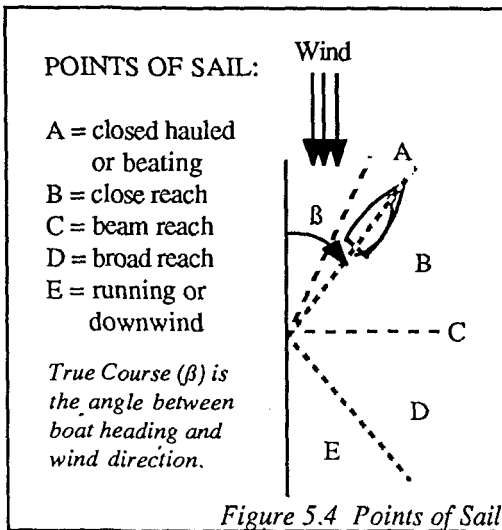
Figure 5.3a is a list of sailboat-related symbols used in this document. True course(β) is the direction of the sailboat in relation to the wind. Figure 5.3b lists the different variables related to boat heading and wind. They differ in their range of heading values. True course is a single value [0°-360°]. Point of sail is a range of headings within 0° to 180°. And tack is either port (β is 0° to 180°) or starboard (β is 180° to 360°). (See Appendix B for more information regarding sailing terminology).

Boat Heading and Wind gives:	
	True Course
	Point of Sail
	Tack

Figure 5.3b

When listing ECHIDNA structures and methods, a modified form of the ECHIDNA CLP language is used for brevity. In the text of the document, references to actual names of schemas and variables are capitalized.

5.2.2 Points of Sail.



The point of sail relates the heading of the boat in relation to the wind. This is useful for reasoning because it provides a consistent view of the sailboat regardless of the true wind direction. Where the true course angle is the absolute angle (from 0° to 360°) between the boat heading and the wind direction, the point of sail is a wider range of headings. For example, if a boat is heading into the wind, the boat's *point of sail* is either close-haul or close reach which means that the boat must be on a *true course angle* between

30° to 50° for a close haul or 50° to 80° for a close reach. The points of sail are the same for both tacks (i.e. starboard and port). Figure 5.4 shows a boat on a port tack (i.e. wind going across the left side or port side of the boat and sail on the right side).

5.2.3 Sailing Actions.

Different sailing *actions* are found at each point of sail. In a *real* boat, sailing actions are tacks and jibes (and maybe an occasional man-overboard exercise). Otherwise, the boat moves under continuous adjustment of the heading and sails. In the NSAIL system, adjustments to the boat heading (i.e. heading up or bearing away without tacking or jibing) are modelled as actions that change the point of sail: beat, reach and run.

5.2.4 Predicting Sailboat Performance.

Predicting the actual performance of the sailboat under given conditions is quite complex. The NSAIL model, since its focus is not on modelling the underlying physics of sailing, relies on experimental data to compute boat speed. The reader is referred to Appendix B for more details on performance calculations.

5.3 NSAIL SYSTEM ARCHITECTURE.

As shown in Figure 5.5, NSAIL consists of two components: the ECHIDNA reasoning component and the sailing application interface which is the display component. Each sailboat object consists of three agents for planning, reaction and perception. Each

agent is represented in both components, and they communicate with their counterpart in the other component through ECHIDNA's XOP. Control within NSAIL is shared between the two components, and there is a close link between the two components of each agent. The sailing animation occurs in two stages for each boat. The first stage is the planning stage. The planning starts before any activity actually begins, and may be re-invoked during the execution of the plan. This corresponds to the onshore pre-race strategic planning that may occur in a real sailing scenario. The second stage is the animation of the plan generated by the planning agent.

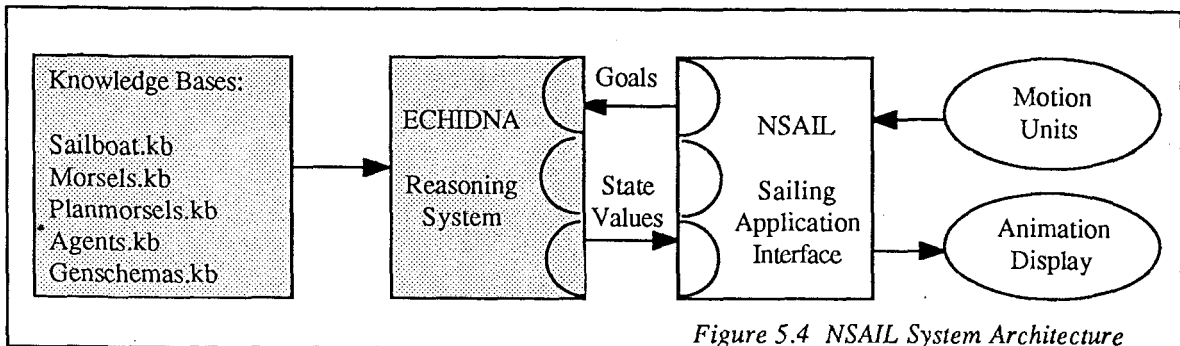


Figure 5.4 NSAIL System Architecture

5.4 WORLD REPRESENTATION.

5.4.1 Objects.

Presently, NSAIL can create sailboat objects, markers and generic static and moving objects. Sailboat objects are the only intelligent objects modelled in NSAIL (i.e. with associated knowledge agents). These objects are used to create the *set* and *characters* for the animation piece. There is also a simple rectangular object to represent land mass. The position and other information about the object is input through the NSAIL user interface.

There is really only one type of sailboat in NSAIL: the hobiecat. The hobiecat is a catamaran-style dingy with a single main sail. Despite the suggestion of its appearance, the hobiecat object symbolizes the typical sailboat. For purposes of rendering, different display models can be attached to the hobiecat object. On the reasoning side, the hobiecat object has a list of object morsels stored in the `Boat_Morsel_List` variable (see Figure 5.6).

```

schema hobiecat
{
  objectNum  Boat_Id.
  boatstate  Boat_State.
  angle      Main_Sail_Angle.
  worldPoint Position.

  agent      Tactician.
  agent      Captain.
  agent      Lookout.

  Boat_Morsel_List.
}
  
```

Figure 5.6 Hobiecat Schema

There is currently no special knowledge in the system pertaining solely to hobiecats¹⁷ (e.g. a hobiecat cannot sail as close to the wind as a monohull).

The BOATSTATE schema (see Figure 5.7) contains variables that are used to define the configuration of the boat. The variables are related to each other through the boat heading and the wind direction. These two variables determine the values for the other variables in the boat state.

```

schema boastate
{
  direction      Heading.
  tacks          Tack.
  pointOfSail    POS.
  trueCourse     TrueCourse.
  operation      Action.
  mfInterval     Modifier.
}

```

Figure 5.7 BoatState Schema

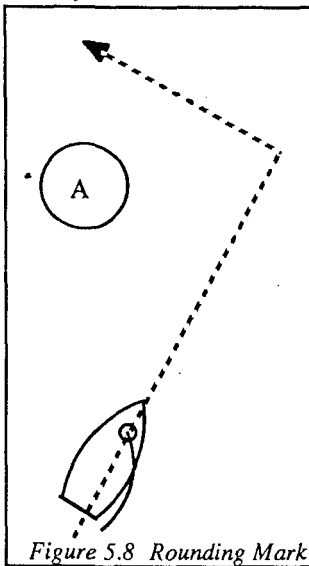


Figure 5.8 Rounding Mark

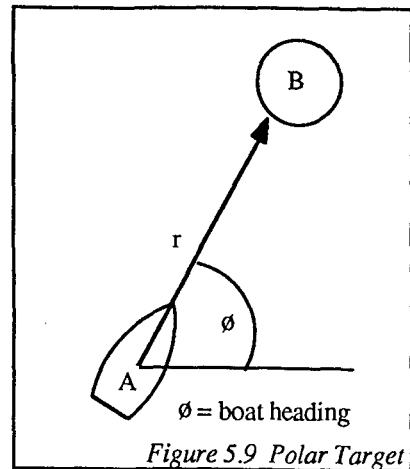
Markers are used in a sailboat race to identify the segment or *legs* of the race course. The sailor knows that the boat must go around the mark in a particular manner to qualify for completion of the sailing leg. This is called *rounding a mark* as shown in Figure 5.8. Marks must be rounded on the side dictated by the race committee (e.g. round to port). Handling of marker objects requires planning since they infer a more complicated context-dependent relationship than other static objects floating in the water. Context-dependency is a reference to the characteristics of the agents controlling the boat and the current goal being pursued. For example, if the sailboat is not in a race situation, the marker object can be handled as a generic static object. It serves as a *navigational object* only, constraining the boat heading to avoid collision with the marker object.

There are other navigational objects found in a sailing domain. In NSAIL, navigational objects define a desired heading relationship between itself and sailboats. For example, a channel buoy means to stay on the starboard side when it is flashing green, etc. The `land_mass` object in NSAIL also can be seen as a navigational object. The heading of the boat should avoid beaching the sailboat! As with other navigational objects, the relationship between boat and `land_mass` object is represented with a `polarTarget` schema (see Figure 5.9).

¹⁷This however can be added at a later date by defining a new kind of sailboat object in ECHIDNA and the application interface. However, no experimental information is available for accurately modelling the different sailboats. Furthermore, the interest is in animation as opposed to simulation.

5.4.2 Targets.

The heading constraints implied when encountering a navigational object are represented with a two-dimensional polarTarget schema instance. In Figure 5.9, a polar target schema is applied to constrain the heading (ϕ) of sailboat A to reach object B. The polarTarget schema simply provides a modular manner for encoding heading relationships between two objects in ECHIDNA's CLP language.



The position of A is unified with the origin of a polar target instance, and the position of B is unified with the endpoint. In the object_morsel for a navigational object, appropriate settings will be defined to relate the location of the endPoint to its own location.

5.4.3 Motion Units.

Motion units are on the display component side of NSAIL and form the physical layer of sailing motion. They are used to implement the actions to be

animated. Though there may be a one-to-one correspondence between actions and motion units, NSAIL has four basic motion units: sail_forward, sail_curved, start_boat and stop_boat. These four motion units are used to implement the NSAIL goal actions listed in Figure 5.10. Starting and stopping a boat is described by Marchaj [Marchaj88] as a path with changes in velocity along the path. This is compatible with the NSAIL implementation of motion units.

NSAIL GOAL ACTIONS:

```
tackOp = { tackToPort, tackToStarboard }.
jibeOp = { jibeToPort, jibeToStarboard }.
beatOp = { portBeat, starboardBeat }.
reachOp = { portReach, starboardReach }.
runOp = { portRun, starboardRun }.
roundOp = { roundMarkToPort, roundMarkToStarboard }.
```

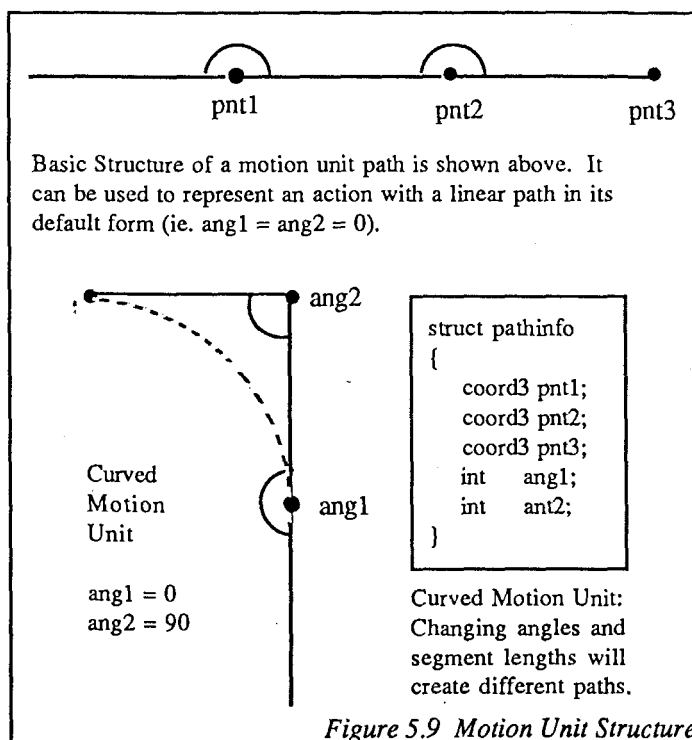
Figure 5.10 NSAIL Goal Actions

In NSAIL, the motion unit is a path with changes in heading along the path. The basic structure defines the path of the motion unit using three points and two angles (as shown by the PATHINFO structure in Figure 5.11). The points are relative to the origin of the movement (i.e. the world point in which the current action began). The angles are relative to the original boat heading when the action started. They indicate the change to the boat heading at the corresponding point (i.e. adjustment by tiller to steer the boat in a new direction). Each point delimits a segment for optional special processing. All motion units continue moving in the direction of the final heading after point *ptn3*. This is required to

handle the transformation from one action to the next. The points in the motion unit also identify when an action may be interrupted to execute an evasive manoeuvre to avoid an obstacle. For example, in a tack or a jibe action, point *pnt1* is the point of no return. The portion of the curved_path action between points *pnt1* and point *pnt3* cannot be interrupted, permitting representation of actual sailing motion cannot be interrupted once started. There is a point in a tack or jibe where there is little control over the boat heading if aversive steering is required. For example, when the sail is luffing^B during a tack as it crosses the wind, there is little power in the sail and hence little maneuverability. In such situations, if the captain fails to check for oncoming boats beforehand, collision may be unavoidable.

The NSAIL *sail_forward* motion unit handles sailing at a steady heading for the different points of sail. It may be interrupted at any time and there is no alteration to the boat heading along the path. There are no special processing segments for *sail_forward* so all points in the path are set to (0,0,0). The boat therefore, when executing a *sail_forward* direction, progresses forward in its present boat heading until otherwise directed (i.e. target reached or obstacle encountered).

The *sail_curved* motion unit is used to implement a tack or jibe action. A curve is fitted¹⁸ to the points in the path of the motion unit. The boat follows the defined path during execution of the action. The distance to the first point refers to the optional preparation portion of the action. For a tack, this is the phase where the speed and heading of the boat is adjusted in anticipation of the tack. A boat must be moving at a certain minimum speed in order to propel the boat across the wind.

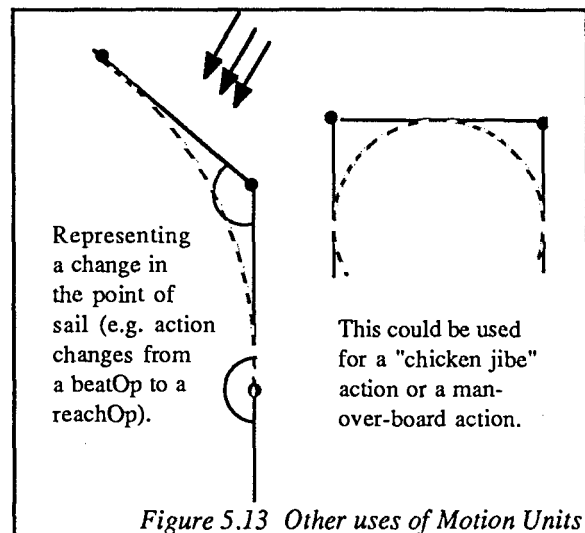
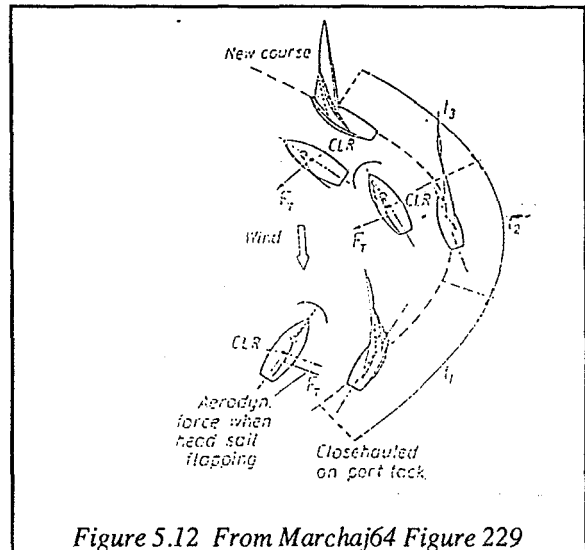


¹⁸Currently, NSAIL is using CTB curve interpolation [Kochanek84]. However, CTB curves pass through all keypoints so point *pnt2* has to be adjusted accordingly. Curve fitting should be done with Hermite or Bezier curves since they do not pass through all the given keypoints.

In a *sail_curved* motion, the heading is altered appropriately and gradually as the action progresses. This is done by the helmsperson at the tiller or wheel. How quickly the heading should be changed, and how quickly the boat responds, is dependent on current environmental conditions. For example, in strong winds, jibing the boat can occur very fast. In lighter winds, tacks must be slow and steady, especially on a hobiecat. Otherwise momentum is lost and the boat may not be able to complete the tack, leaving the boat *in irons* (i.e. luffing straight into the wind and basically going nowhere or possibly backwards!). This is represented by adjusting the relative distance of the points in the motion unit path.

Analysis of data from wind tunnel experiments and sailing races [Marchaj64], support the division of tack actions into distinct segments (see Figure 5.12). There are also fluctuations in the boat speed due to variation in the aero-hydrodynamic forces on the sailboat. The sail angle must be adjusted at different points along the path as well. This information can be encoded at the different points of the motion unit path structure. The changes in heading and sail angle during the execution of the motion are not propagated to ECHIDNA until the motion is in a steady state (i.e. after point *pnt3*). ECHIDNA does not need to reason about these changes prior to this point.

The *sail_curved* motion unit can also be used to animate a more realistic transfer from one action to another. An example is the changing of the boat direction to move from one point of sail to another, or one action to another (see Figure 5.13). If the heading is to be different at point *pnt2*, the captain actually starts turning the boat at point *pnt1*, and the boat arrives at its new heading at point *pnt3*.



6.0 NSAIL KNOWLEDGE MORSELS.

6.1 OVERVIEW

The NSAIL reasoning component specifies the motion of a sailboat at the behavioural level. The behavioural level represents the reasoning process of the intelligent entity controlling the mainsail angle and boat heading. The intelligent entity is modelled by three NSAIL agents: the tactician, captain and lookout. The character of these agents is captured in ECHIDNA by three kinds of knowledge morsels: boat morsels, reaction morsels and planning morsels (see Figure 6.1.).

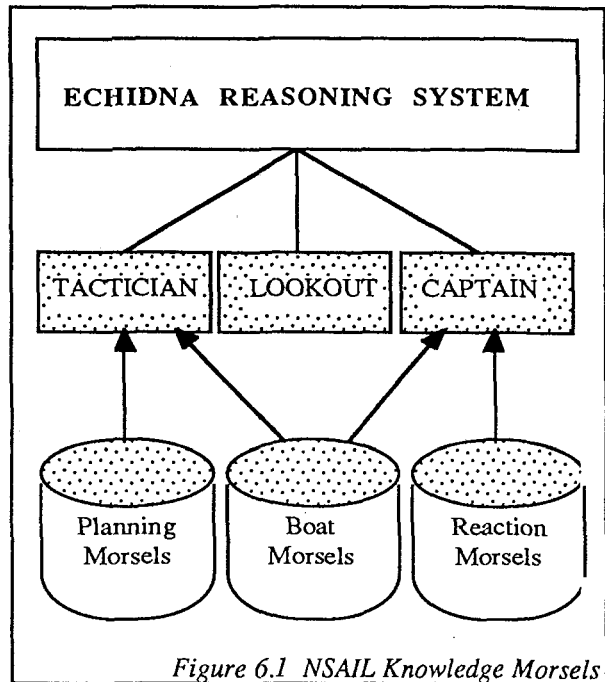


Figure 6.1 NSAIL Knowledge Morsels

6.2 BOAT MORSELS

When a sailboat is instantiated, its agents are given a list of morsels to reflect the knowledge of the world in which they interact. In NSAIL, the tactician and the captain must have knowledge about how a sailboat works in order to accomplish their respective tasks. This knowledge is encoded in boat morsels for reasoning within ECHIDNA.

Boat morsels are object morsels that relate the boat state to the wind direction¹⁹. There are three types of boat morsels: point of sail, tack and sail. Each morsel type, identified by a unique id, has the basic structure shown in Figure 6.2.

SAILBOAT MORSELS (boatMorsel):

```

pointOfSailMorsel
tackMorsel
sailMorsel
  
```

Figure 6.2a Sailboat Morsels

```

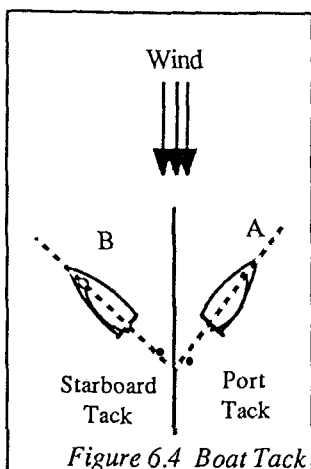
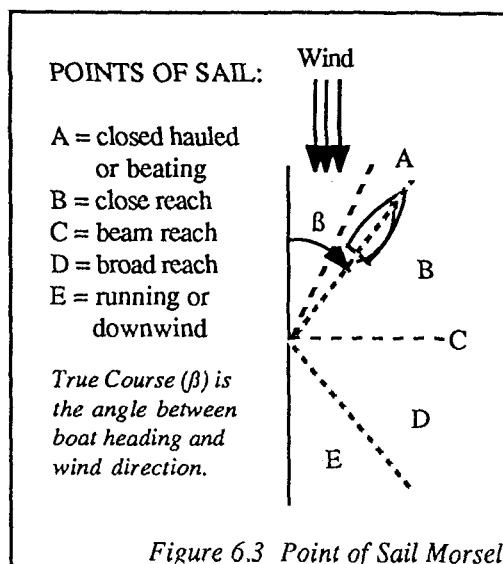
schema boatMorsel
{
  boatstate BState.
  boatMorsel( boatstate B ) :-
    BState = B.
  apply :- constraints.
}

bind( ID, boatstate B, boatMorsel M ) :-
  M isa boatMorsel(B).
  
```

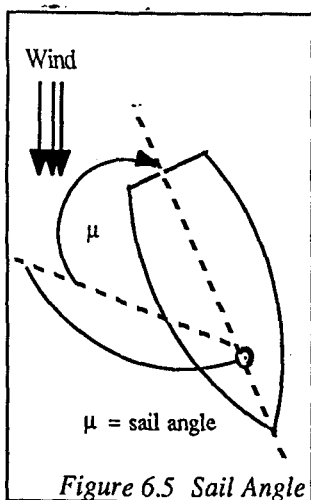
Figure 6.2b Boat Morsel Schema

¹⁹A sailboat moves under the influence of both aerodynamic and hydrodynamic forces (i.e. wind and water). To reduce complexity, hydrodynamic forces (e.g. waves, tides, etc.) are not considered in NSAIL.

The POINT_OF_SAIL_MORSEL schema defines the interdependency between the boat state variables and the boat's point of sail. POS (the point of sail variable) is determined by the true course (β) of the boat. True course angle is given as a single value relating wind direction and boat heading. The point of sail defines the relationship between the wind direction and boat heading with a range of values for true course. If the true course is between 90° and 165° off the wind (area D in Figure 6.3), the point of sail is then a broad reach. The points of sail basically describes the major relationships between the boat heading and the wind direction (i.e. upwind, across the wind or downwind). Sailing behaviour differs significantly at these general points. Thus, the point of sail morsel also identifies particular sailing actions for each point of sail.



The TACK_MORSEL schema defines the relationship between boat tack and true course (β). The true course is already related to boat heading in the point of sail morsel. A boat is on a *starboard tack* if β is 180° to 360° and a *port tack* if β is 0° to 180° (see Figure 6.4). The TACK_MORSEL schema also relates the boat tack to appropriate sailing actions (e.g. port beat or starboard beat).



The SAIL_MORSEL schema (see Figure 6.5) defines the relationship between the point of sail and the main sail angle (μ). When the boat is sailing upwind the sail is pulled in close to the centerline. As it *bears away* (i.e. as it turns downwind from close hauled to running), the sail should be adjusted accordingly to maximize boat performance. The adjustment for bearing away is usually to let the sail out so the sail angle is increased for optimal wind flow and driving force (see Appendix B). However, the sail angle is not wholly derived from boat heading and wind direction as with tack and point of sail. The SAIL_MORSEL schema only offers the normal mapping between the sail angle and point of sail.

Range of values for sail angle where $\mu=0^\circ$ means that the sail is sheeted in close to the centreline of the boat and the point of sail is usually close hauled.

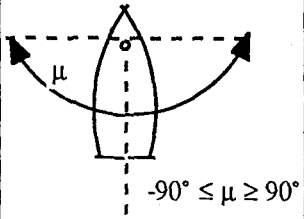


Figure 6.6 Sail Angle Range

The implementation agent may choose instead to set the sail angle differently for various reasons (e.g. to depower the sail and reduce boat speed). It may choose any value for the sail angle μ where $-90^\circ \leq \mu \leq 90^\circ$ (see Figure 6.6). The agent may not know the best sail position which is dependent on many factors (see Appendix B). Since the boat speed is affected by the sail angle, it can be used to model the inexperienced sailor. Thus, the sail morsel includes a null APPLY method which allows the sail angle to vary independently of the constraints stated in the morsel. Excluding the boat morsel from the morsel list leaves the sail setting totally in the hands of the implementation agent.

The boat morsels essentially relate the boat state variables to the boat heading and the wind direction. If either one changes, the boat state will be automatically updated by the backtracking process. Thus, the constraints in the boat morsels identify all the possible sailboat configurations.

6.3 REACTION MORSELS.

Reaction morsels (see Figure 6.7) define the relationships between objects in the NSAIL domain, mainly for collision avoidance. All sailboats have morsels to handle generic physical objects, both static and moving, in the domain. For each object P_i in a sailboat S 's panic box (where $1 \leq i \leq n$ and n is the number objects in the panic box), a reaction morsel R_i is created to relate the state variables of S and P_i .

staticMorsel (navigational)
movingMorsel (navigational)
landMorsel (navigational)
rightOfWayMorsel (sailboat)

Figure 6.7 Reaction Morsels

The reaction morsel for a generic static object P_i imposes a constraint on the boat heading (ϕ) to pass on either side of P_i . Since boat heading is an interval variable in ECHIDNA, the OR operator is not available. Therefore, the APPLY method for the reaction morsel will have two clauses, one for each side of P_i . As the position of sailboat S is updated during the animation phase, the avoidance headings (angles π and Π in Figure 6.8) may be modified due to unpredictable boat behaviour (i.e. it may stray off course). In such cases, the persistent reaction morsel will adjust the heading in response to existing conditions.

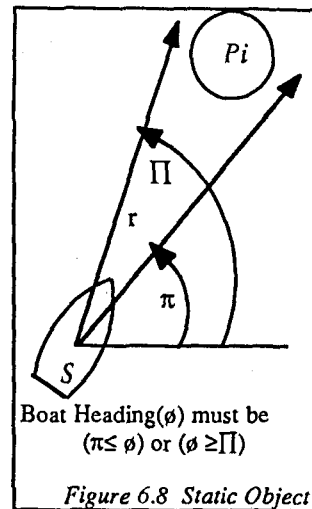
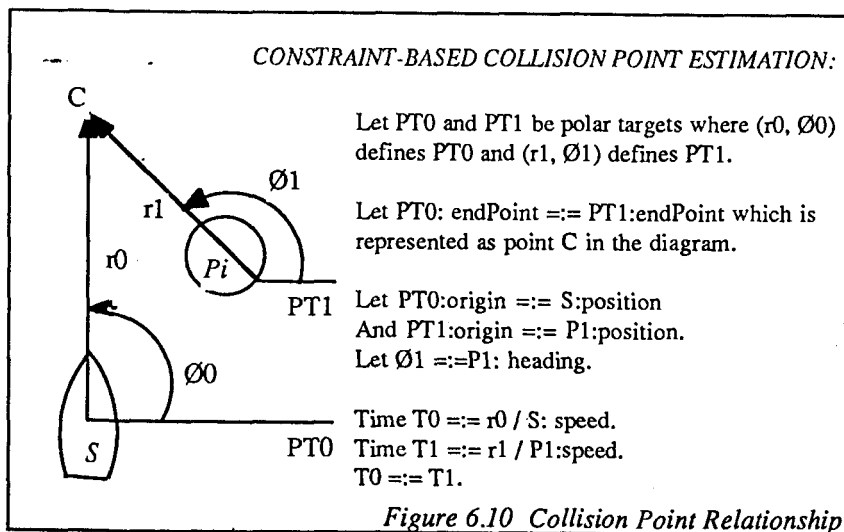
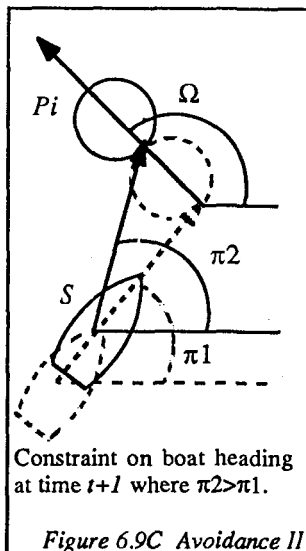
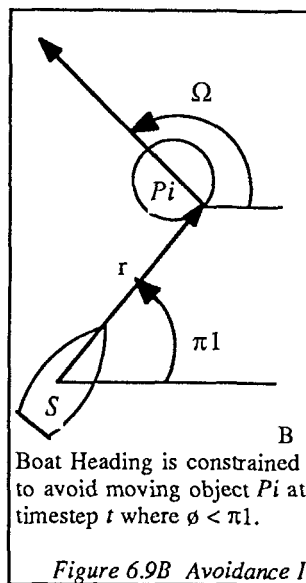
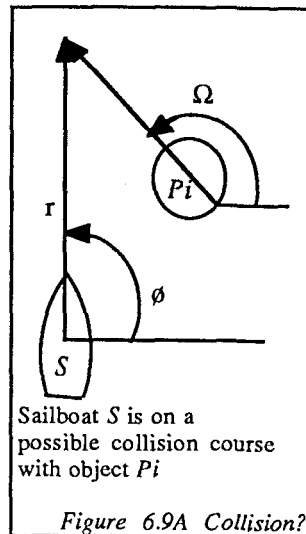


Figure 6.8 Static Object

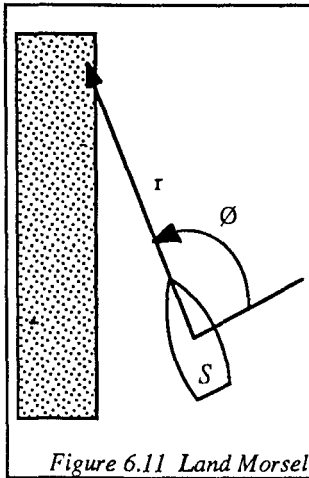
If sailboat S is on a possible collision course with an unidentified moving object P_i (see Figure 6.9A), P_i will be in the panic box for S . The relationship between S and P_i is defined by the reaction morsel for generic moving objects.

The generic moving object morsel uses a polar target to maintain the relationship between the two objects. The position of the sailboat S is bound to the origin of the polar target, and the position of object P_i is bound to the endpoint. Thus, the polar angle (π_1) constrains S 's heading to pass behind P_i (see Figure 6.9B). As P_i moves away, the constraint on the sailboat heading relaxes (as in Figure 6.9C) until P_i exits S 's panic box.

Aiming to sail behind object P_i is the most cautious way of avoiding collision. The method works fine for objects that are fast and close. For objects that are further away, it is more appropriate to estimate the collision point and constrain the boat heading for sailboat S to avoid the collision point. This is encoded in ECHIDNA with multiple polar targets (see Figure 6.10). As depicted, a collision is when the same point (C) is reached at the same time by two objects. Polar targets $PT0$ and $PT1$ in Figure 6.10 are effectively encoding two vectors, representing S and P_i respectively, to identify a moving collision point over a period of time.



In Figure 6.10, the reaction morsel for handling a moving object P_i now constrains the boat heading to aim below the collision point C (i.e. boat heading $\phi < \phi_0$ or $\phi > \phi_0$ where ϕ_0 is the polar angle for polar target PT_0). The exact heading needed to aim below point C is dependent on the direction of P_i . It is also possible to aim forward of the collision point. If P_i is travelling directly towards or away from S , the boat should head towards either side of P_i as if it was a static object (refer to Figure 6.7).



A land morsel defines the relationship between sailboat S and a land object L . In this case, the constraint is simply that the position of sailboat S not be within the land mass area (see Figure 6.11). For the NSAIL implementation, the land constraint is further simplified to symbolize a shoreline, expressed as a line (i.e. $x=x_0$). Thus, the constraints with L operates on only one coordinate of S 's position in world space. The land morsel schema is introduced in NSAIL to provide an alternative constraint type (i.e. one that does not directly impact the boat heading as most other reaction morsels do).

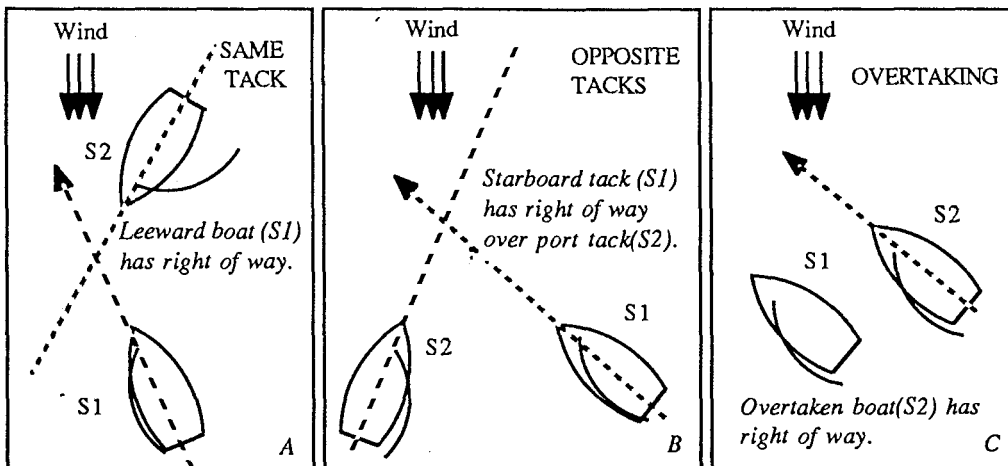


Figure 6.12(a,b,c) Right of way rules for sailing

The right-of-way morsel defines the right of way rules between two sailboats S_1 and S_2 where S_2 is in the panic box of S_1 (see Figure 6.12). There are basic right of way rules²⁰ to handle the three possibilities when two boats meet. For boats on the same tack

²⁰There are actually three general sets of rules for sailing: International, Inland and Racing. The International Rules follow the three basic rules. The Inland Rules include an additional rule: A closehauled boat has the right of way over a free running boat. This is from the days of the square riggers which were more manoeuvrable downwind than closehauled. The Racing Rules allow the opposite tack rule to override the overtaking rule when the situation occurs. That is, a starboard tack boat does not have to stay clear when it is overtaking a port tack boat. This only occurs when sailing downwind and the defense is for the boat being overtaken to tack. [Colgate73]

(Figure 6.12A), the leeward boat (i.e. the one further downwind) has the right of way. For boats on opposite tacks, the starboard boat has the right of way (Figure 6.12B). In overtaking situations, the boat being overtaken has the right of way (Figure 6.12C). In racing situations, there are additional rules and exceptions to the basic rules which can be integrated as part of the tactician's strategy to gain an advantage over another boat. For example, when rounding a mark, the inside boat will try to create an overlap^B in order to gain the right of way. (See Appendix B) Only the basic rules are presently included in the right of way morsel.

In NSAIL, if sailboat S_1 has the right of way, it can hold its course by maintaining a right of way relationship with sailboat S_2 . Otherwise S_1 must establish a relationship with S_2 for collision avoidance. These are separate APPLY clauses (see Figure 6.13) in the morsel for backtracking (i.e. only one clause can be consistent with the boat states of the two boats). For the same tack rule (Figure 6.12B), the leeward relationship is provided through an external method for ease of computation. For the third rule (Figure 6.12C), object S_1 is overtaking object S_2 if heading of $S_1 =:=$ heading of S_2 , and the speed of $S_1 >$ speed S_2 .

Right of way morsel:

```
apply :- S1 is on starboard tack
apply :- S1 is leeward of S2
apply :- S1 is being overtaken by S2
apply :- S1 must avoid S2
apply :- S1 must reduce speed
```

Figure 6.13 Right of Way Morsel

The collision avoidance clause finds the collision point C (as for a generic moving object), and constrains the boat heading of S_1 to be forward or aft of C . It is left to other strategic constraints to determine a more definite heading (e.g. an aggressive agent would pass ahead if the boat velocity is improved). Another technique for collision avoidance is to slow the boat down by adjusting the sail (i.e. to "spill" wind to depower the sail). This is included as the final option for collision avoidance in the right of way reaction morsel.

6.4 PLAN MORSELS.

Plan morsels (see Figure 6.14) define the relationship between adjacent plan nodes in a strategic plan. Specifically, plan morsels relate goal nodes to their precondition nodes. In the proposed planner, for each plan node i in the plan list, node i is the precondition node for node $i+1$ where $0 < i < n-1$ and n is the number of nodes. Plan node n also represents a top-level goal from the NSAIL application interface.

PLAN MORSELS:

```
tackOperationMorsel
jibeOperationMorsel
beatMorsel
reachMorsel
runMorsel
roundMarkToPortMorsel
```

Figure 6.14 Plan Morsels

NSAIL implements only a small subset of possible plan goals (e.g. round mark to port, tacks, jibes and point of sail variables). This subset, however, is sufficient for the exploratory nature of the NSAIL application. The most challenging point of sail, beating upwind, is covered in this planning subset.

Recall that a plan node (see Figure 6.15) has an action (represented by the GOALOP variable), plan state and target point. The plan state is the same as a boat state, and they are unified during plan execution after the action is done. Note that the heading of the precondition node is towards the target of the goal node.

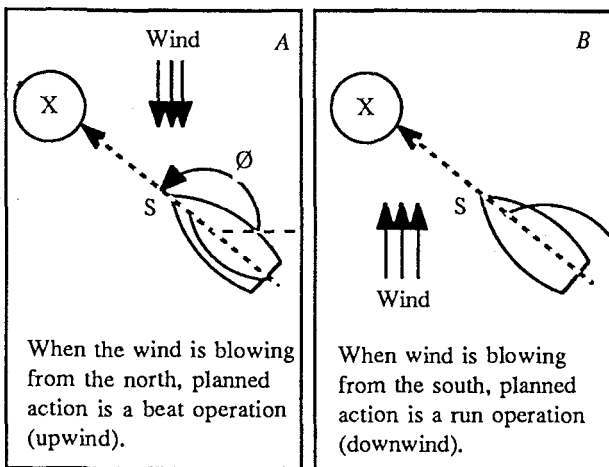
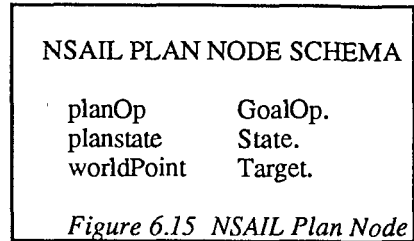
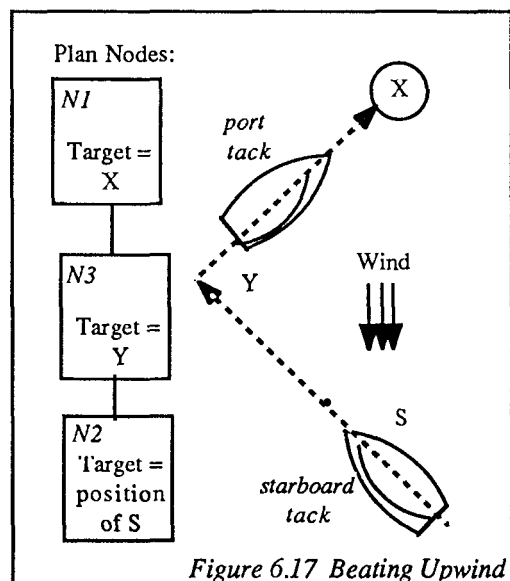


Figure 6.16 Determining Action for a Target Goal

A basic goal for a sailboat is to sail to a given location in the world space. The top-level goal plan node specifies only the target location X . A polar target T is then used to relate X to the position of the sailboat S . The boat heading is unified with the polar angle \emptyset . The endpoint of T is unified with X and the origin of T is unified with the position of S . (See part A of Figure 6.16) The current boat state of S is bound to the plan state. Now when values are assigned to the two plan nodes, the polar target T will constrain the assignment to ensure that the boat heads towards the target X .

Simply using a polar target (as in Figure 6.16) to find the appropriate heading for a target goal works well except when the target X is too far upwind of the sailboat S . It is impossible to sail directly upwind towards X . Some types of sailboats may be able to head as close as 30° off the wind, but on average, most boats can only get as close as 40° [Marchaj88]. In such circumstances, a series of tacks must be made to reach the upwind target. Thus, the beat plan morsel has a clause that constrains the target of the precondition node (e.g. node $N3$ in Figure 6.17)



to be the midpoint of the windward distance between the sailboat *S* and target *X*. This is the most optimal point for a tack if there are no interfering obstacles. Therefore, any new plan node is first heuristically constrained to be halfway between its two neighboring plan nodes. New plan nodes are added only when there is no other option to satisfy the goal node *N1* with precondition node *N2*. A new plan node *N3* is added as the new precondition node for the goal node *N1*. Node *N1* will constrain the heading at node *N3* to aim towards target *X*, and node *N3* will constrain the heading at node *N2* towards target *Y*. The location of target *Y* may be further constrained by node *N3* and by other morsels (i.e. boat morsels and reaction morsels for static objects) that are part of the planning process.

A tack operation means to change the boat tack from starboard to port (or vice-versa) by turning the bow of the boat across the wind. The `tackOperation` morsel (see Figure 6.18) relates the plan states of the goal node and its precondition node. Note that the tack operation requires a heading change of 90°. In this manner, the target variable of node *N3* in Figure 6.17 will be constrained to an appropriate location. A jibe operation also changes the boat tack, but while sailing downwind so that the stern turns across the wind.

Let N1 be the goal plan node
Let N2 be the precondition node for N1

```
tackOperationMorsel:
  N1:tack(tacks T1),
  N2:tack(tacks T2),
  T1 != T2,
  N1:pos(pointOfSail P1),
  N2:pos(pointOfSail P2),
  P1 := sailingUpwind _,
  P2 := sailingUpwind _,
  N1:heading(direction H1),
  N2:heading(direction H2),
  H1 := H2 ± 90.
```

Figure 6.18 Tack Operation Morsel

In a sailing race, rounding a mark to port means to go around a race mark with the mark on the port side of the boat (see Figure 6.19). The round mark goal does not really correspond to a specific action. It is the action taken at its precondition node that actually moves the boat around the mark when the plan is executed. The `roundMarkToPort` morsel strategically sets the target and plan variables for the precondition node. Rounding a mark is a very crucial aspect of the race. This is where all the exciting strategy happens in a race:

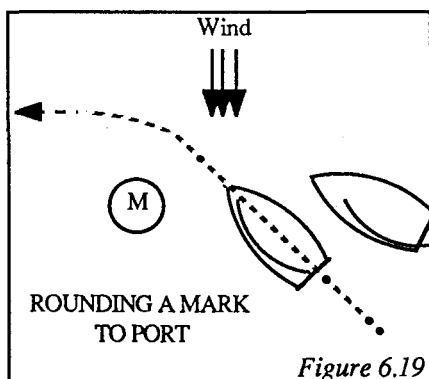


Figure 6.19

Always approach a mark so that you can round it without having to tack in the process. If you are rounding marks to port, a starboard tack approach is generally best; you have the right of way over other boats and can round without tacking. The most desirable position at the mark is usually on the inside line. If you can establish an overlap and round closest to the mark you will be the windward boat after rounding, in good position to inflict other boats with your backwind, blanket area and wake! Don't waste time tacking immediately after rounding the mark. It's best to maintain course on the same tack; this way you'll have the jump on boats trailing close behind you. Closely monitor the strategy of your nearest opponents. When they alter course or tack, you can too and still stay in front with a clear air advantage. [Grubb79]

Such simple words of strategy can be quite complicated to implement! Currently, in NSAIL, the only strategic heuristic applied is to be on a starboard tack when rounding a mark to port. The next easiest strategic heuristic to implement in this framework is to be as close to the mark as possible. The other points mentioned in the excerpt [Grubb79] are too complicated to incorporate in NSAIL.

7.0 NSAIL Control and Animation.

7.1 OVERVIEW

As shown in Figure 7.1, control in NSAIL is passed from agent to agent for different functions. The user designs the environment and specifies high-level goals. For each sailboat, its high-level goal, plan morsels and boat morsels are used by the tactician to produce a plan list²¹. The plan list, boat morsels and reaction morsels constrain the motion of the sailboat during plan execution by the captain. Changes in each sailboat's dynamic environment are stored in their panic box by the lookout and used to trigger replanning when necessary.

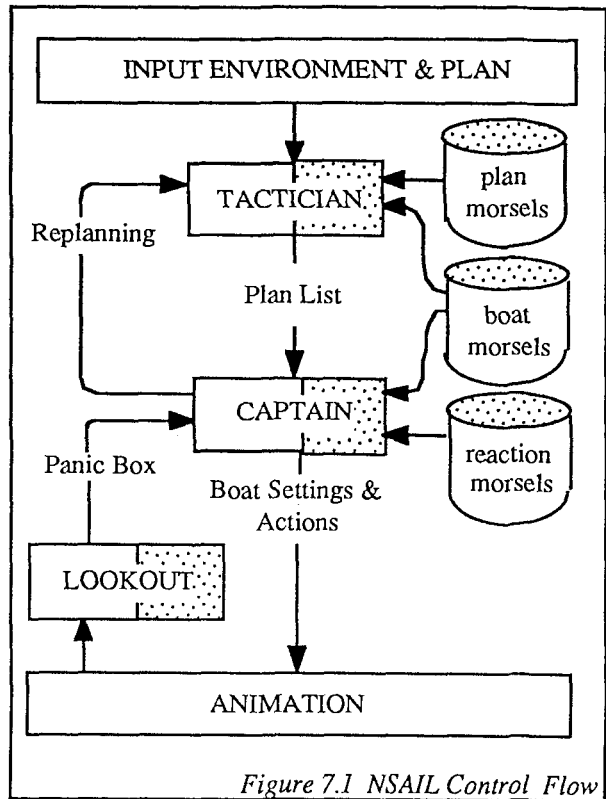


Figure 7.1 NSAIL Control Flow

7.2 KNOWLEDGE AGENTS.

The behaviour of each NSAIL agent follows the model proposed in the conceptual design where the tactician is the planning agent, the captain is the implementation agent and the lookout is the perception agent (see Figure 7.2). The knowledge specific to the NSAIL implementation is captured in the knowledge morsels used by the agents. The boat morsels and plan morsels are used by the tactician to produce a plan which relates the goal of the boat to its initial position.

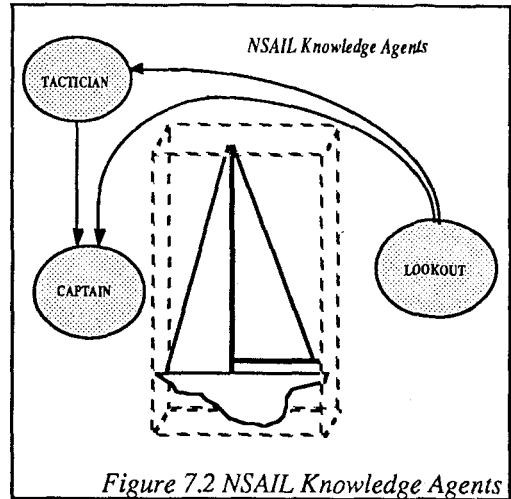


Figure 7.2 NSAIL Knowledge Agents

A top-level goal (e.g. round mark to port at marker X) is specified when the sailboat is created by the user via the NSAIL user interface. The planning process is also controlled by the display component through the tactician. To control the planning

²¹Originally, the intent was to do a high-level plan of the race course. Hence, the inclusion of the "onshore" kind of planning expected of the tactician.

process, the tactician issues the following top-level ECHIDNA goals shown in Figure 7.3. Note that for each planning agent, the details of steps two and three follow the basic planning algorithm in Figure 4.9.

- | | |
|-------------------|--|
| 1) SETUP_GOAL: | Binds and applies boat and plan morsels in anticipation of the planning activity. This is done for all sailboats before progressing to the next planning task. |
| 2) PLAN_POSITION: | For all sailboats, assign initial position of the boat to the first plan node. |
| 3) ASSIGN_PSTATE: | Assigns values to the plan node variables (i.e. target, plan state and action). |
| 4) PLANLIST: | Application captain binds each plan node in the generated plan to the boat state for plan execution. |

Figure 7.3 Top Level Planning Goals

Once planning is completed, the captain relates the first plan node to the boat state in preparation for the animation phase. During execution of the plan, replanning occurs for a particular sailboat if the state of the world is not consistent with its goals. Therefore, the top-level plan goal will fail and the display component side of the tactician will initiate the replanning process with the updated location of the sailboat. The first step is undo all the previous goals in order to deal with the non-monotonic environment.

- | | |
|------|---|
| 1) | Undo ASSIGN_PLAN_STATE. |
| 2) | Undo PLAN_POSITION. |
| 3) | Redo PLAN_POSITION goal with current position of sailboat |
| --4) | Redo ASSIGN_PLAN_STATE |
| 5) | PLANLIST - bound to new plan node |

Figure 7.4 Replanning

7.3 CONTROL HIERARCHY

The implementation of the plan requires a hierarchical control structure. The animation cycle (see Figure 4.13) is repeated for each object at each time step. Each object in turn cycles each of their associated agents through a series of steps for initialization and screen update. Each agent on the display component side performs a series of ECHIDNA goals to apply morsel constraints and assign state values to be used

for calculating the next position of the object. Thus, the agent cycles through its lists of morsels and requests a series of goals. For example, during the animation cycle, when a

hobiecat object assigns values to the boat state, it requests that its agents perform an assignment task. Each agent sends an ASSIGN goal to each morsel in its morsel_list. This is required for binding, assignment and application of morsels. This is necessary because some operations must be done for

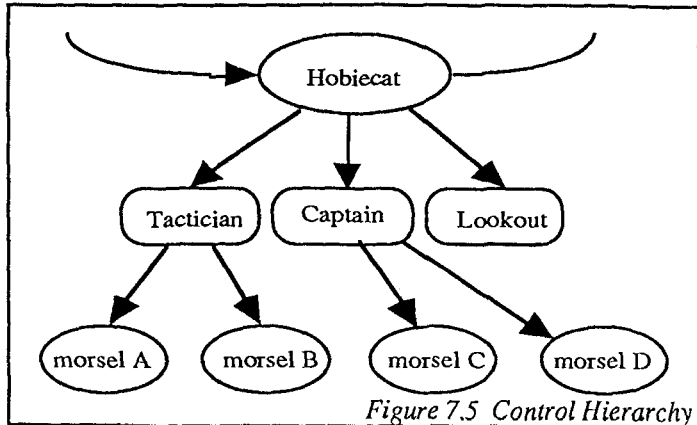


Figure 7.5 Control Hierarchy

all objects at one time. For example, all objects must apply the constraints in their morsels before any variable can start assigning values (i.e. all constraints must first be applied to the object population).

7.4 THE NSAIL USER INTERFACE.

The NSAIL user interface is interactive and is intended to produce real-time control of the 3D animation. The animation process follows the given steps: 1) design the layout for the sailing environment (i.e. positions of static objects and land masses); 2) describe the characteristics of a sailboat by selecting knowledge morsels for each of the knowledge agents; 3) instantiate multiple sailboats with specified locations, headings and goals; 4) set the wind characteristics; and 5) initiate the animation display and watch the motion evolve. Environmental conditions can be changed as the animation proceeds. The sailboats, through the reasoning component, will automatically respond to such environmental changes (i.e. the wind) as the animator manipulates the values of environment factors. Boats can be deleted and added, goals can be changed, and the layout can be rearranged to control the animation. This is the ideal situation.

The current NSAIL interface is implemented in an object-oriented language (GNU G++) to be consistent with the object-oriented design of ECHIDNA. Both components of NSAIL, the reasoning the display components, run on Silicon Graphics Iris Indigo workstations, and communicate through the XOP. The user interface, shown in Figure 7.3, is constructed with the FORMS graphical user interface toolkit. A useful testing feature is a FORMS panel for entering ECHIDNA goals from the display side.

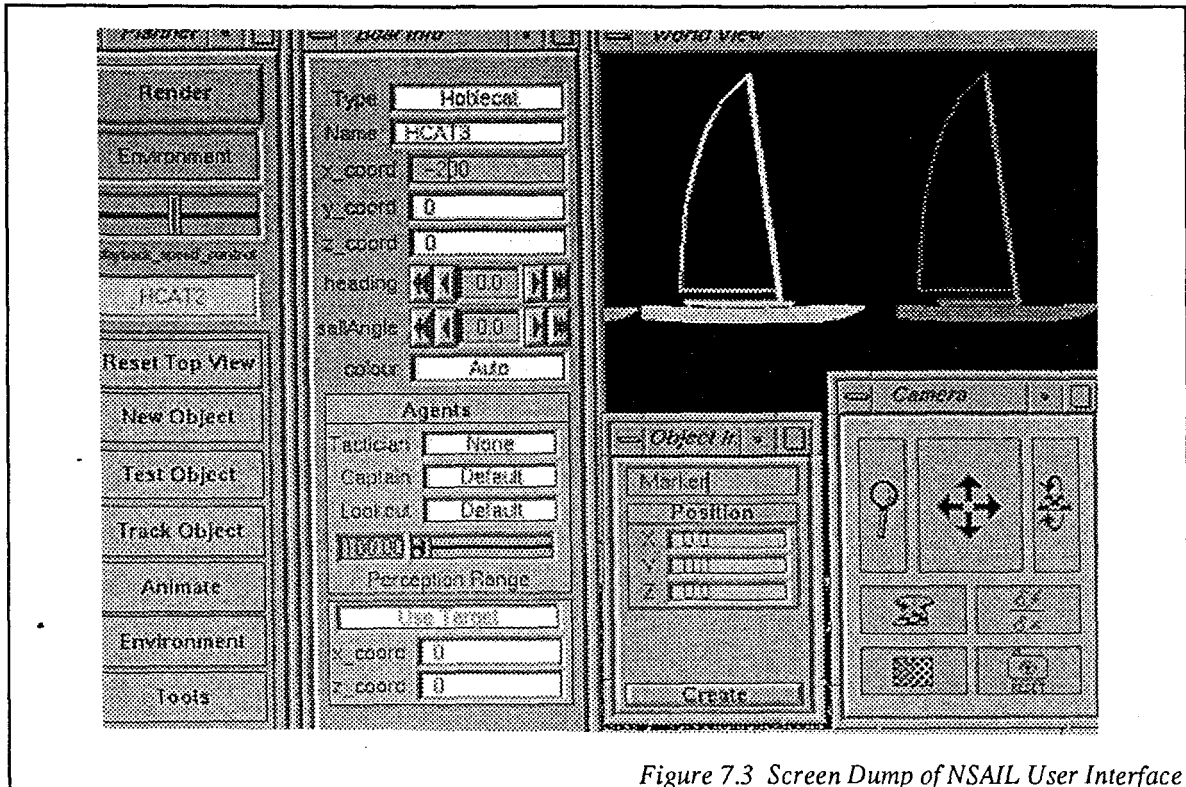


Figure 7.3 Screen Dump of NSAIL User Interface

7.4.1 Components of the NSAIL User Interface:

The following is a brief description of the current version of the NSAIL interface.

1. *sailboat creation* A sailboat is created by entering boat information via the BOATINFO panel. Position, boat heading and other parameters may be set initially. There are choices for different kinds of agents (e.g. it is possible to assign only a captain agent to sailboat). Different agents dictate different behaviour. The *perception range* determines the panic box size for the boat.

There are currently only two goals: 1) sail to a user-specified target; and 2) round mark to port. The target goal is used mainly for testing.
2. *object creation* It is possible to create a marker, a generic static object and a land mass object. A simple environment is defined by creating and positioning static objects.
3. *animation control* 3D viewing control and tracking is available. Any sailboat may be selected for tracking. The rate of display may be changed, and playback of stored data is being implemented.
4. *wind control* The wind direction and speed may be changed during the animation. The sailboats in the scene should respond appropriately to the wind changes.

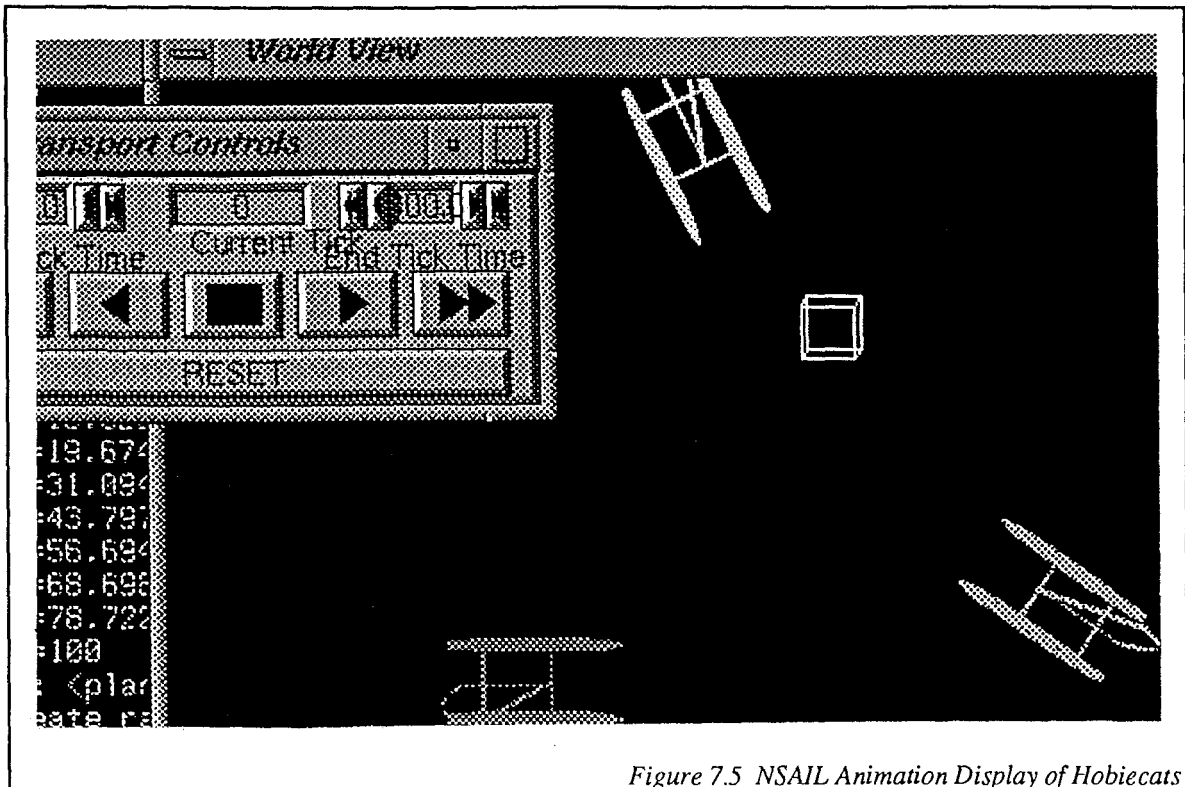


Figure 7.5 NSAIL Animation Display of Hobicats

The NSAIL user interface provides an interactive way to control the animation of the sailboats. It also explored the possibility of real-time animation control where the animator can view the animation at the same rate as the final display (i.e. at a rate of 30 frames per second for video). However, for more than two boats doing anything more than reactive motion (i.e. using only reaction morsels), the hobicats jump more than sail due to long periods between screen updates. Currently, it is more useful to view the processing in a step wise manner, requesting the system to calculate changes at specified time intervals. Providing a batch facility for reasoning and computing changes in positions of objects is also a possible option.

With current version of NSAIL, an animator can create a simple environment with multiple sailboats, and communicate with the reasoning agents during animation of the sailing scene. The hobicats are capable of determining their appropriate state in relation to the wind direction. They can set their heading towards a mark, but confusion arises once they reach their mark (see Figure 7.5). There is a tendency to keep going around the mark. The engineering of the planning agent and its interaction with the implementation agent is in progress. The initial results are promising. Incorporating some of the findings from the discussion to follow will improve the efficiency of the implementation.

8.0 DISCUSSION.

8.1 THE EVOLUTION.

The development of the proposed integrated reasoning and animation framework evolved with the ECHIDNA constraint-based reasoning system. Both components continue to evolve. In this chapter, issues related to the current state of the system are discussed. The implementation issues address the details of the ECHIDNA constraint logic programming language and their application within the proposed animation framework. The representation issues deal with particulars of the NSAIL implementation. An evaluation of the overall conceptual approach is also presented.



Figure 8.1 An Echidna Tack

8.2 IMPLEMENTATION ISSUES.

8.2.1 Interval Variables.

In computer animation, the world space is a 3D Cartesian coordinate system. Eventually the motion of animated objects refers to changes in coordinate positions from frame to frame. In the initial framework, the world space is restricted to a 2D coordinate system. Since the two components of the proposed system are closely integrated via the ECHIDNA XOP, *similar* representations of the world space are used within the application program and ECHIDNA. Thus, in ECHIDNA, positions of objects are points in a 2D coordinate system, and participate in constraints. The initial intent was to represent each coordinate as a discrete variable.

However, for efficiency, the cardinality of a discrete constraint in ECHIDNA is limited to 1,000,000. The cardinality of a discrete constraint is defined as the product of the cardinality of the variables that participate in a constraint. The cardinality of a discrete bound variable is the number of elements in its domain. Thus, the cardinality of a coordinate (i.e. x or y) is restricted to 1000 if represented as a discrete variable. This is too low of a granularity for the application side. Furthermore, even at a much lower

cardinality, the slow down in execution time is significant for simple constraints. Given these considerations, interval variables are used to represent coordinate values and other parameters in the NSAIL environment. This also serves as study of interval variables since they are not widely used in other ECHIDNA applications.

Interval variables^A provide the efficiency and the range desired for representation of world parameters. However, interval variables cannot be used with the inequality constraint operator (\neq). This requires the definition of *masking* intervals, using the NOTIN primitive, to create the desired negation of an interval variable. This can be difficult and introduce additional choice points^{22,A} in the solution. Also, the disjunction of interval constraints (i.e. OR) has not yet been implemented, and only constant values can be used with the NOTIN primitive. When applied in a discrete constraint, OR can eliminate a choice point from the solution. Finally, on the display component side, it is the midpoint of the interval variable that is used in calculations.

The ITERATE structure is another primitive only available for discrete variables. ITERATE relates the same constraint to an array of values. By consolidating a series of constraints into a single node in the resolvent tree, a contribution is made towards greater processing efficiency. It also improves readability of the code for an ECHIDNA method.

8.2.2 Intervals and Precision.

The precision of an interval variable is the degree to which it is refined. An interval variable has an associated precision value which is set by default to the global precision of the ECHIDNA session. The precision value indicates the number of times an interval variable is split. Each split introduces a binary choice point into the resolvent tree. As mentioned, introducing choice points effects the backtracking process. The higher the precision, the greater the number of choice points and the greater the effect on backtracking. Thus, a higher precision trades off efficiency. However, the degree of impact on efficiency is context-dependent. If a large part of the tree is pruned by other constraints, the impact is reduced. On the other hand, if the precision is too low, unpredictable results occur.

The PRECISION primitive is intended for use in incremental design problems. By varying the global precision at successive stages of a design problem, the level of detail

²²See [Sidebottom92b] for a more detailed explanation of choice points.

can be refined [Sidebottom92b]. For most other applications, including NSAIL, the precision is usually specified at the start of an ECHIDNA session and left unchanged. However, the global precision value is context-sensitive, and it is not clear what the optimal setting should be for different kinds of problems. In NSAIL, the significant use of interval variables enforces the need for careful selection of the global precision.

8.2.3 Interval Segmentation.

Trigonometric functions require careful consideration when used in a constraint expression with interval variables. Because of the cyclic nature of these functions, an interval variable may be broken into many segments, increasing the greater complexity in the constraint network. Thus, the bounds on the interval should be economized to eliminate unnecessary choice points from segmentation. That is, the bounds of an interval need to be wide enough that valid solutions are not eliminated, possibly causing undue failure. While on the other hand, they need to be narrow for efficiency when related with trigonometric primitives.

8.2.4 Proper Constraints and Choice Points.

A *proper* constraint does not introduce a choice point into the resolvent tree. A proper constraint is a single constraint expression or a method with a single clause. Proper constraints serve to actively narrow the search space once they are applied. Heuristics²³ in NSAIL, because of the nature of sailing strategies, introduces choice points. Thus, it is best to apply proper constraints first to reduce the search space, and then apply heuristics afterwards. Maximizing the number of proper constraints and minimizing the number of choice points lead to greater efficiency.

In the implementation of NSAIL, there is no clear separation between proper constraints and heuristics. Given the organization of knowledge into morsels, proper constraints are distributed throughout the database, and applied in an ad hoc manner. To maximize their effectiveness, a distinction should be made in each morsel between proper constraints and other constraints such as heuristics. Proper constraints for each morsel should then be collectively applied first.

²³Heuristics need not always introduce a choice point in ECHIDNA. However, in NSAIL, heuristics are viewed as options to be tried to maximize boat performance. If the option is not consistent with other constraints, then another option is tried or nothing at all. This is encoded as methods with multiple clauses, introducing choice points into the resolvent tree.

8.2.5 Use of Preprocessed Knowledge.

Preprocessed knowledge are known mappings between values (as shown in Figure 8.2). Before introducing the ELEMENT primitive, indexing into an array with the ARG primitive required a ground value. Thus, mappings between values are encoded

as a method with multiple clauses, corresponding to a choice point (see Figure 8.2B). The ELEMENT primitive permits unground indices, and sets up a constraint between the index and array elements, avoiding addition of another choice point.

A:	
element({2..3}, <100, 400, 600, 700>, X). X = { 400, 600 }.	
B:	
P(X,Y).	
P(1, 100).	Normal encoding without ELEMENT because X is not ground so cannot use ARG primitive.
P(2, 400).	
P(3, 600).	
P(4, 700).	

Figure 8.2 The ELEMENT Primitive

8.3 REPRESENTATION ISSUES.

8.3.1 The World Space.

The representation of the world space for the reasoning module needs to be more sophisticated. A strict mapping of the coordinate system for animation, the *animation space*, to a coordinate space in ECHIDNA, the *reasoning space*, is not a good approach. Currently, in NSAIL, there is a *translation mapping* between the animation space and the reasoning space. The translation mapping simply maps the location of the objects in the animation space to a corresponding location in the reasoning space. The reasoning space uses only the first quadrant of a 2D coordinate system where coordinate values are all positive. This is necessary to avoid problems with interval arithmetic when relating an interval variable with negative values to the exponentiation primitive²⁴. Of course, the animation space can be adjusted to be the same, but it is not necessary, or optimal, to have the same granularity of representation in ECHIDNA and the application code.

Given this view of the world, the constraints must also be ordered in hierarchies, or be capable of operating on different levels of abstraction. In the hierarchical coordinate space shown in Figure 8.3, a relationship between A and B need not refer to a relationship between two points; it could also be a relationship between two grid areas. Thus, at

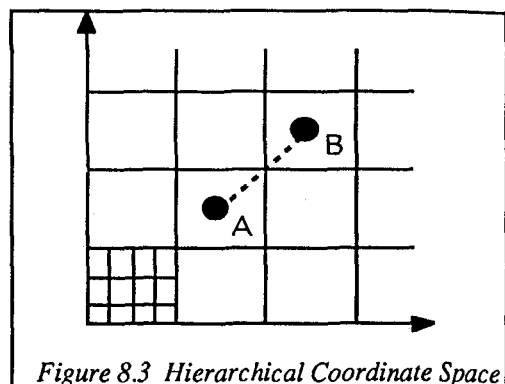


Figure 8.3 Hierarchical Coordinate Space

²⁴Exponentiation cannot be used in an expression with interval variables that have negative values in their range.

one level, a sailboat may be heading towards a particular target point while at another level, it is aiming towards a larger area. The NSAIL knowledge morsels currently relate only the *point positions* of objects in the world. This is conducive to the level of granularity needed for geometric computations on the application side, but may not be the right level of abstraction for reasoning. A hierarchical world coordinate space is an alternative representation that is compatible with views of the hierarchical organization of the brain and mind [Rummelhart76, Simon69].

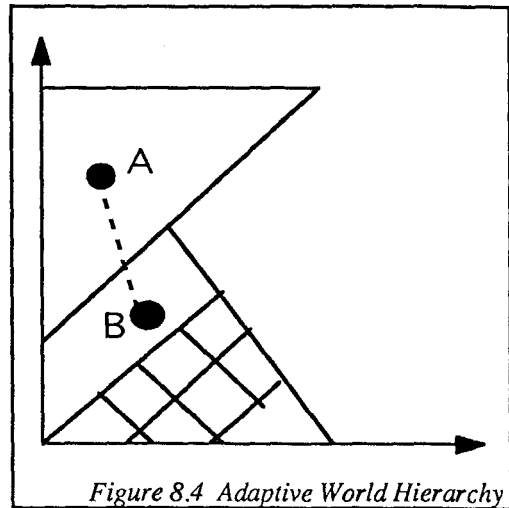


Figure 8.4 Adaptive World Hierarchy

The hierarchical structure may also be adapted to the environment (see Figure 8.4), possibly as seen from a particular agent's view. In a sailing environment, areas of open water may be grouped together as one *chunk* in the reasoning space. Areas with a denser population of static objects are further subdivided into smaller chunks. Perhaps a *fish-eye* or *variable zoom* view is appropriate [Schaffer93]. Given a hierarchical representation of the world space, the intelligent objects can reason and respond to different levels of stimuli from the environment. The agents must also be able to handle the greater complexity of hierarchical objects and knowledge bases. It is not obvious how to apply ECHIDNA, or any constraint logic programming language, to represent and reason about a world that exists simultaneously at different levels of abstraction.

8.3.2 Use of External Methods.

In NSAIL, external methods are used for various purposes: to compute the contents of a sailboat's panic box (i.e. all objects in its range of perception that are on a colliding course with the sailboat); to determine leewardness (i.e. given two boats, which boat is downwind of the other); and to calculate new positions based on the active motion unit for a sailboat. These computations can be done with relative ease outside of the reasoning system. The information returned is then used as part of the reasoning process to relate other variables. This procedure improved efficiency by pursuing optimal use of the capabilities of an integrated system.

Brown in his discussion of problem solving by computation verse search space

exploration, emphasizes the need to recognize when one approach is more approach than another:

If during diagnostic reasoning one needs to know the exact value of pressure in the reactor chamber, if one has an equation that can be evaluated for it, and if one has all the information needed to evaluate the equation, then it is silly to use problem-space exploratory methods. On the other hand, for a number of problems such as diagnosis and design in the general case, the underlying spaces can be very large, and solution algorithms of restricted complexity are generally not available. This is when AI methods are appropriate. [Brown89 page 4].

Thus, within an integrated framework, it is important, from a practical point of view, to recognize when to apply two different kinds of approaches: reasoning with declarative knowledge and procedural computations through external methods. Though there seems to be a stronger dependence on declarative knowledge, in living organisms, it is usually agreed that intelligent machines need both procedural and declarative knowledge [Genesereth87, Rich83].

8.3.3 Handling Change.

In NSAIL, the top-level position goal for each sailboat is redone at each time step. This is necessary to handle change in a world which is represented by a nonmonotonic formalism. In the current version of ECHIDNA's object-oriented CLP, there is no mechanism to address the *nonmonotonic state change problem* [Havens92]. In ECHIDNA, a schema instance variable may be repeatedly refined towards a ground value through propagation of constraints and unification. Its domain can only be elaborated by backtracking through nondeterministic methods. Once ground, it cannot be reassigned another value. Thus, it is not possible to change instance variables representing the sailboat parameters, once they are ground. Thus, at each time step, the top-level goal for boat position must be undone and redone with the new position to relate the change in position to other instance variables.

However, *thrashing* occurs during backtracking if new constraints are added after doing SPLITS and INDOMAINS to assign values to instance variables. Undoing a constraint and redoing it is the same as adding a new constraint. Thus, when the position goal is redone, it adds new constraints, causing thrashing to occur. To avoid this problem, all the goals invoking SPLITS and INDOMAINS must be undone at the top level before the new position goal can be reapplied (see Chapter 7). When a goal is undone,

proper constraints are persistent, but backtracking will occur for choice points. In NSAIL, the degree of backtracking is very high because of the organization of knowledge as morsels, the use of interval variables and the heuristic nature of the sailing knowledge.

A clearer separation of proper constraints and choice points may improve the performance of NSAIL, but a better mechanism is needed for handling state change in the proposed framework. This may be provided by primitives in ECHIDNA for handling nonmonotonicity. Animation is not a monotonic process; it deals with changes over time. ECHIDNA must be able to handle not just changes in boat variables but with time as well. Advancing time cannot be done with undoing the assignment to the time variable.

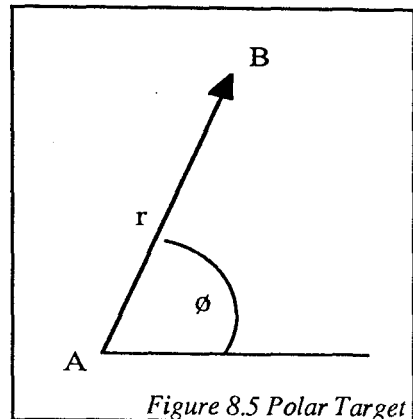


Figure 8.5 Polar Target

In earlier explorations of handling time for NSAIL, time was represented as an instance variable related to position. Perhaps a feasible approach would be to include time as another variable in the problem and rely more on external methods for computationally intensive relationships²⁵. As a trivial example, a polar target (see Figure 8.5) is used in the knowledge morsels to relate two objects in terms of an angle ϕ and a radius r . Object A is a moving object and object B is a static object. If r is then constrained together with time (e.g. by an external method or simply $\text{time} ::= \text{distance} / \text{speed}$), the position of A can be determined by assigning all values of its domain to time variable t (i.e. by using the INDOMAIN and FAIL primitives).

In NSAIL, the positions would actually be determined by external methods because of the computations needed in the motion units on the application side, but this will eliminate the need for repeated undoing of top level goals by the application module. Using the FAIL primitive still means backtracking but less so since only one constraint with the INDOMAIN primitive on t is failed. However, this option for advancing time was not used in order to explore the potential for real-time animation controlled from the display component of the system. This option must also be considered with the additional complexity of morsels and plan nodes. Furthermore, reasoning about multiple timesteps would not be possible since only one time step is valid at any instant.

²⁵Even if the gain in performance is insignificant, there may be a gain in ease of programming.

8.3.4 Planning and Plan Execution.

The planner in the proposed framework is relatively simple and based loosely on the NOAH backward planner [Rich83]. To achieve a certain goal, the planner starts with two nodes, the goal plan node and the plan node with the initial state of the sailboat. If it is impossible to achieve the goal with just two nodes (i.e. there is no single action to take the boat from the initial state to the final state), a new plan node is added. The constraints are applied to the new set of plan nodes and an attempt is made to assign actions and state values to this new set of plan nodes. Thus, in effect, the planner is basically redoing the planning process at each plan step. In NSAIL, the planner is only used to determine the sailboat path for sailing upwind on a beat (i.e. when the target is directly upwind of the sailboat and multiple tacks are needed). The number of nodes required is relatively small. However, in general, the planning approach may be improved by using heuristics about goals and actions for specific domains to estimate the number of nodes needed at each planning step.

A more difficult planning task in NSAIL is the recognition of goal satisfaction. A plan node is completed when the sailboat reaches a target destination, completes the planned action and assumes the post-action state in preparation for the next plan step. Thus, goal achievement is a continuous activity. It not a single state, but a series of states that must be recognized over time. In NSAIL, this recognition is performed by setting appropriate flags through top-level goals from the implementation agent on the application side. At each time step, a flag variable in ECHIDNA is unified at the top level with a flag value indicating which state of the plan is currently being executed by the sailboat²⁶. This is similar to approaches in traditional animation where there are stages of anticipation, plan action, secondary action and follow through [Lasseter87].

8.4 EVALUATION OF APPROACH.

8.4.1 Constraint Logic Programming Environment.

Representing knowledge in a constraint logic programming language provides an efficient formalism that is sound and complete (i.e. robust). More importantly, the declarative nature of the representation lends itself to intelligent problem solving, including solving the problem of sailing around race markers. This kind of behaviour is

²⁶Control structures are scheduled for implementation in ECHIDNA using the XOP.

characterized by heuristically-based searches through a solution space [Brown89] which is facilitated by a logic programming language (e.g. inferencing and backtracking capabilities). In regards to NSAIL, it is possible to implement the sailboat animation system with a procedural approach adapting priority allocation mechanisms to resolve conflicting behavioural rules (as in [Reynolds87]), but the complexity of interaction between the rules will quickly become apparent. Assigning priorities to behavioural rules would be an expert task.

In NSAIL, the organization of knowledge in terms of morsels and agents favours the use of heuristics, and unfortunately, decreases the benefits of using constraints to reduce the search space. However, modifications discussed earlier should improve performance. It is not clear at this point whether intelligent backtracking influences the efficiency of the NSAIL reasoning process. In a strongly interconnected solution space, intelligent backtracking has less impact. A strongly interconnected space means that all variables are intertwined through constraints and constraint propagation. Thus, when a failure occurs, it is impossible to determine the source of the error for intelligent backtracking.

8.4.2 Constraint Visualization Tools.

The difficulty in determining the effectiveness of constraint logic programming and intelligent backtracking identifies a lack of development tools for the knowledge engineer. This is a common deficiency with declarative knowledge approaches in general. It is hard to follow the flow of the problem resolution process. Tools for visualization of constraints would be of great benefit. However, it is not obvious what should be visualized? And perhaps what should be visualized is a domain -dependent task. For example, in NSAIL, an attempt was made to visualize the determination of boat heading but the degree of backtracking rendered the task useless.

A constraint visualization tool would be extremely helpful, but may have to be domain dependent since relationships are meaningful, not as abstractions in a resolvent tree or search tree, but as reflections of their domain. For example, the relationships between sailboats and the wind would not be useful if visualized as a complex network of choice points. Designing a general visualization tool would be a challenging task. Perhaps what is needed is a toolkit of visualization primitives for variables and constraints that will enable the knowledge designer to create domain specific visualizers.

8.4.3 The Proposed Animation Framework.

The modularity and potential for diversity is an attractive feature of the proposed framework for an integrated reasoning and animation system. The benefits of an object-oriented paradigm is found in both the reasoning and animation display components. Though the two components are tightly integrated, an attempt is made to clearly identify the XOP communication points through the structure of the knowledge agents. This is a step towards a *plug-in* system organization, where it would be possible to attach different knowledge morsels and agents for objects on the display component side that have a particular form and motion. For example, the sailing behaviour in NSAIL may be *plugged into* an animation interface for human figures where the boat heading is mapped to the orientation of the figure and the sail angle to a limb joint. Specialized *human sailing movements* can be created, using keyframing, dynamics or any other technique available in the application, to correspond to the sailing motion in the reasoning system.

The object-oriented constraint logic programming language within the ECHIDNA reasoning shell is a powerful formalism for knowledge representation. It allows the model of the world to be generalized, providing inheritance in a manner that supports a kind of default reasoning. For example, all physical objects in the world are classified as static or moving objects. Therefore, if there is no specific morsel for defining the relationship between particular objects, interaction can be deferred to the general case. This is a reflection of the learning process. Unless there is some acquired experience with a specific object, it will be handled as a member of a generic class in which it has been classified. For example, a navigational buoy that indicates a boat should pass on the starboard side will be treated as a generic static object unless the agent *knows* that the buoy signifies that a particular behaviour must be followed. Navigational buoys can be metaphors for general objects in animation environments.

8.4.4 Planning.

Planning is a continuous process. Initially, in NSAIL, a tactician may plot a course before the start of a race but the planning process really continues for the duration of the race. This implies more complicated interactions between the planning agent and the implementation agent. The right of way rules control reactive behaviour, and the boat responds according to these rules when it encounters another boat. However, once the boat yields or changes course in response to its relationship with the other boat, it must then replan to find its way back to the previous heading. That is, in the course of its

reactive behaviour, it changes its heading completely, overriding the planned heading.

Execution of the plan is a perception-driven process. Though it may be possible to include better simulation of vision in the proposed framework, there needs to be a higher level of organization pertaining to the information collected from the environment. In NSAIL, perception is a list of objects in the lookout's range of sensitivity. If there are too many objects in that range, it may not find a state consistent with all the objects in this range (i.e. there is no single direction that is not on a collision heading with another boat). Reducing the range of perception will decrease the number of objects that must be handled simultaneously. In this manner, the sailboat maneuvers around the closest object before dealing with the next. However, grouping objects into clusters would be indicative of more intelligent behaviour by the captain. An avoidance course can then be planned to go around clusters of objects. For example, in sailing, a possible strategy at the start of a race is to favour the tack with the least number of boats.

9.0 CONCLUSIONS.

One primary objective of this thesis was to explore the use of a constraint-based reasoning system for behavioural motion specification and control in computer animation. In conjunction with the exploration, and as a contribution, this thesis work developed a conceptual framework for an integrated reasoning and animation system. The proposed framework served as the basis for NSAIL, a reasoning and animation system for sailing behaviour. The development of the NSAIL testbed continues to be an incremental process, feeding back into the design of conceptual framework.

The ECHIDNA object-oriented constraint-based reasoning system provides a necessary high-level formalism for knowledge representation with a logic programming language. Furthermore, the object-oriented features of ECHIDNA are well suited for the conceptual framework. This initial investigation identified some key points in building an integrated behavioural animation system, and systems in general, using a declarative formalism. A significant observation is that there needs to be a better understanding of the kinds of problems suitable for a declarative representation. The general problem of reasoning and animation of behaviour is conducive to a declarative representation, and to a constraint-based logic programming representation. However, within the general problem, there are subproblems that are more compatible with a procedural approach. In further development of the proposed framework, it is important to choose an appropriate representation at multiple levels of implementation.

At the detailed level of implementation, it is important to choose the appropriate representation of variables in the reasoning domain. Intervals can provide efficiency but they should be used selectively due to some of the difficulties in using intervals (i.e. precision, segmentation, and compatibility with representation on the application display component side). An understanding of constraints, constraint propagation and intelligent backtracking contributes to better knowledge base designs. Constraints only help to reduce the solution space if applied properly. These details of implementation play a significant role in the efficiency of constraint propagation and intelligent backtracking.

The difficulties in developing a system in a constraint-based logic programming environment include the lack of tools for the designers of the knowledge bases. In nontrivial implementations, it is a very challenging task to follow the flow of constraint

propagation and backtracking. A constraint visualization tool would be extremely helpful but may have to be domain dependent since relationships are meaningful, not as abstractions in a resolvent tree or search tree, but as a reflection of their domain. However, it is not clear what is an appropriate representation for visualization of constraint domains.

One key point that has risen is the importance of world representation. Not only is the proper representation needed for the declarative reasoning component, but the selected representation must be compatible with the application display component. What should the link between the two abstraction be? It no longer makes sense to have a one to one mapping. An appealing possibility is a hierarchical representation of the world space, the constraints within the morsels and the planning process. This is conducive to findings in other fields of research including psychology and neural networks [Rummelhart76]. Since the concern of behavioural motion animation is human behaviour, research in such areas can contribute towards the development of a better system integrating reasoning and animation for future work.

Future work includes the application of the proposed reasoning and animation approach to the animation of human group movement (e.g. pedestrians at a traffic intersection). The motion units can be created in the *LifeForms* movement specification system [Calvert93]. The reasoning component will be given the task of determining when a particular motion unit should be activated. For example, the system would reason about when it would be safe to cross the street in a pedestrian crosswalk scenario. There are many other domains to investigate, and the organization of the knowledge bases and system architecture will facilitate the reuse of existing implementations.

In conclusion, the work in this thesis provided insight into the use of the ECHIDNA constraint-based reasoning system, presented a conceptual framework for an integrated reasoning and animation system for behavioural control, and completed work towards pilot implementation of NSAIL, an animation interface for sailing behaviour.

Have a hobie day!

Appendix A: GENERAL TERMINOLOGY.

The following is an alphabetically ordered list of terms used in this thesis. The terms are from computer animation and artificial intelligence (particularly knowledge representation and logic). The terms are defined in the context of this thesis.

- Agent** An agent is a schema instance in the reasoning component of the proposed system that is capable of reasoning about the environment or the state of its associated object. Agents are associated with intelligent objects and are assigned knowledge morsels which define their characters. There is a corresponding representation of the agent on the display component side of the integrated reasoning and animation system. The integrated agent is responsible for controlling the motion of the object.
- Animation** Animation, from the Latin word *animare*, means literally to give life to, or to give *anima* (breathe, soul) or *animus* (spirit, mind, courage)²⁷. Animation is the process of representing changes of movement over time. The illusion of movement is created by presenting a rapid display of successive images at a rate fast enough for the human eye to perceive continuous motion. This rate is 24 frames per second for film and 30 frames per second for video.
- Articulated Body** An articulated body is composed of segments or links (usually rigid) connected at joints, and can be represented by a tree structure. The human body is expressed as an articulated body, often for the purpose of human figure animation in computer graphics.
- Backward Planner** A backward planner is one that plans from the goal node to the initial node. In comparison, a forward planner starts planning from the initial node to the goal node. In the proposed framework, a backward planner is used.
- Blackboard** A blackboard is a shared data structure used to represent knowledge accessible to all domain-specific knowledge modules in an AI program. An example of its use is given in Rich83 on page 278.
- Cardinality** In ECHIDNA, the cardinality of a discrete bound variable is the number of elements in the domain. The cardinality of a discrete constraint is the product of the cardinality of the variables that participate in the constraint. A good guideline is to keep the domain of the variables as small as possible to reduce the amount of time spent in constraint propagation.
- Choice Point** A choice point is a node in the resolvent tree which has alternatives or choices. This may correspond to a method with multiple clauses.

²⁷This is from the Greek word *anemos* which means wind. This seems rather appropriate considering the animation of wind moving sailboats for the proof-of-concept implementation.

- Completeness** See soundness.
- Conceptualization** A conceptualization is a view of a world for the *formalization* of knowledge in a declarative form. A conceptualization includes the objects presumed or hypothesized to exist in the world and their interrelationships.
- Constraint** A constraint is a relationship [Mackworth77]. In the ECHIDNA constraint logic programming language, a constraint is implemented as a persistent data link that supports the flow of information back and forth between variables. That is, it is a bi-directional link.
- CLP.** A CLP is a constraint logic programming language.
- Discrete Variables** A discrete variable has a domain consisting of a set of values from one of the three built-in discrete types supported in ECHIDNA: symbols, integers and reals. A discrete variable is *unbound* if it is simply typed. A discrete variable is *bound* if it is associated with an explicit domain. And a discrete variable is ground if it has exactly one element in its domain.
- Dynamics** This is the study of changes in motion in terms of forces and torques. The classical definition of dynamics is based on Newton's Three Laws of Motion.
- Forward Checking** See search.
- Forward Kinematics** Forward Kinematics involve finding the position of a distal segment of the body given the joint angles of the proximal segments (e.g. finding the position of the foot given the angles of the hip, knee and ankle). See also kinematics and inverse kinematics.
- Frame** A frame is a structure for knowledge representation introduced by Minsky [Minsky75]. A frame consists of slots, each of which describes an attribute, and relationships between the slots. The slots can form a PART_OF hierarchy, and the frames are organized in an IS_A hierarchy. For example, a sailboat IS_A vehicle, and the attribute MAST is PART_OF the rigging.
- Goal-Directed** Specification of motion by assignment of a goal to the entity performing the movement. Goals can be actions, tasks or target destinations. This is the same as task-oriented.
- External Method** An external method is a method that is defined outside of ECHIDNA and is invoked through the XOP which supports communication between the reasoning system and other processes (currently only C++ programs). Invocation of an external method creates a persistent data link between ECHIDNA and the external process, enabling the exchange of information.

Heuristic	A heuristic is a "rule of thumb" that is <i>usually</i> true.
Inference	The process of deriving conclusions from premises.
Interpolation	Interpolation is the process of determining the set of missing values between two known values. For example, in curve fitting, the points forming the curve are found by interpolation through a set of given key points defining the curve. In keyframe animation, interpolation finds the set of inbetween frames between keyframes.
Interval Variables	A real interval variable has an infinite domain defined by a set of intervals. An interval variable is <i>bound</i> if it has an associated explicit interval. It is <i>ground</i> if its domain is refined to the current ground precision.
Inverse Kinematics	Inverse kinematics involve finding all the joint angles of the proximal segments required to produce a particular absolute position of a distal segment (e.g. given the position of the foot, find the joint angles at the ankle, knee and hip, which will place the foot at that position).
Keyframe	A keyframe is a frame in an animation sequence where there is significant change in motion. In computer animation systems, the animator only needs to specify the keyframes, and the system will automatically produce the inbetween frames that are needed to generate realistic motion. These frames are interpolated using splines such as linear, quadratic or cubic splines. This <i>inbetweening</i> process provides smooth transitions between keyframes.
Kinematics	Kinematics is the analysis or description of movements of bodies or parts of bodies in time and space independent of the forces that cause these movements [Wilhelms85]. It involves the calculation of linear and angular displacements, velocities and accelerations. There are two problems, originally from robotics, associated with the kinematics of articulated bodies: the forward kinematics problem and the inverse kinematics problem.
Lookahead	See search.
Panic Box	In the context of this thesis, a panic refers to the list of objects in the environment that are on a collision course with the object associated with the panic box. The panic box is maintained by the perception agent (i.e. the look in the NSAIL application).
Polymorphism	In the object-oriented paradigm, a polymorphic reference is one that can, over time, refer to instances of more than one schema class. Communication between schema instances is attained through message passing. In ECHIDNA, a message is passed to an instance by colon operator and invokes the operation defined by the method

named in the message. With polymorphism, a message may be sent to different instances or classes as defined by the inheritance hierarchy for classes of objects.

- Precondition** Precondition is a term from planning literature referring to conditions required prior to the selection of an operator.
- Predicate Calculus** Given a conceptualization of the world, an appropriate language is needed to formalize knowledge as sentences expressed according to its rules of the grammar. Predicate calculus is such a formal declarative language, with a very precise set of grammatical rules for expressing logical sentences. In *first order* predicate calculus, the variables used refer only to objects in the universe of discourse (i.e. variables cannot be functions or relations). *Second order* predicate calculus includes functions and relations as variables in logical sentences.
- Resolvent Tree** The resolvent tree is a tree of unresolved goals that is maintained by the interpreter in ECHIDNA. The tree begins with the top-level query as the single root node. The leaves of the tree contain the currently unresolved goals at any point in the computation; this set of current goals is called the *resolvent*. See page 78 in Sidebottom92 for a construction example for a resolvent tree.
- Schema** A schema refers to a way of organizing information about commonly occurring patterns of things. The use of schemata, by people as well as programs, exploits the fact that the real world is not random [Rich83]. Other kinds of schemata include scripts for collection of events, stereotypes for collection of characteristics related to people, and rule models for common features shared among a set of rules in a production system.
- Script** A script is a knowledge representation formalism used to describe a common sequence of events such as what happens when one goes to a restaurant.
- Search** VanHentenryck distinguishes the different search strategies as follows: "Generate and test only tests the constraints when a complete assignment has been done. Standard backtracking makes sure that each already assigned variable is consistent with all the other already assigned variables. Forward checking makes sure that each not yet assigned variable has *at least one* consistent value with the already assigned variables. Looking ahead makes sure that each not yet assigned variable has at least one consistent value with *all* the other not yet assigned variables." ([VanHentenryck89] page 17).
- Soundness** Soundness and completeness deals with derivability of an inference procedure (see Nilsson71 p. 54). An inference procedure (a way of making choices and resolutions). Proving derivability is important because it means that the derivation process can be *automated* (i.e.

write a computer program to do it). Soundness means that every sentence in the "database" can be derived from the logical operators of the language. Completeness means that the operators specified for the language can derive all possible sentences found in the database. This is an indication of how powerful the language is.

Unification

Unification is the process of determining whether two expressions can be made identical by appropriate substitution for their variables. Unification is part of the resolution process for inferencing. See Genesereth87 for a complete explanation.

Universe of Discourse

The set of objects about which knowledge is being expressed. That is, the objects in the domain of interest.

XOP

The external object protocol which is the link between ECHIDNA and an application program external to the reasoning system.

APPENDIX B: THE SAILING MODEL

B.1 SAILING TERMINOLOGY

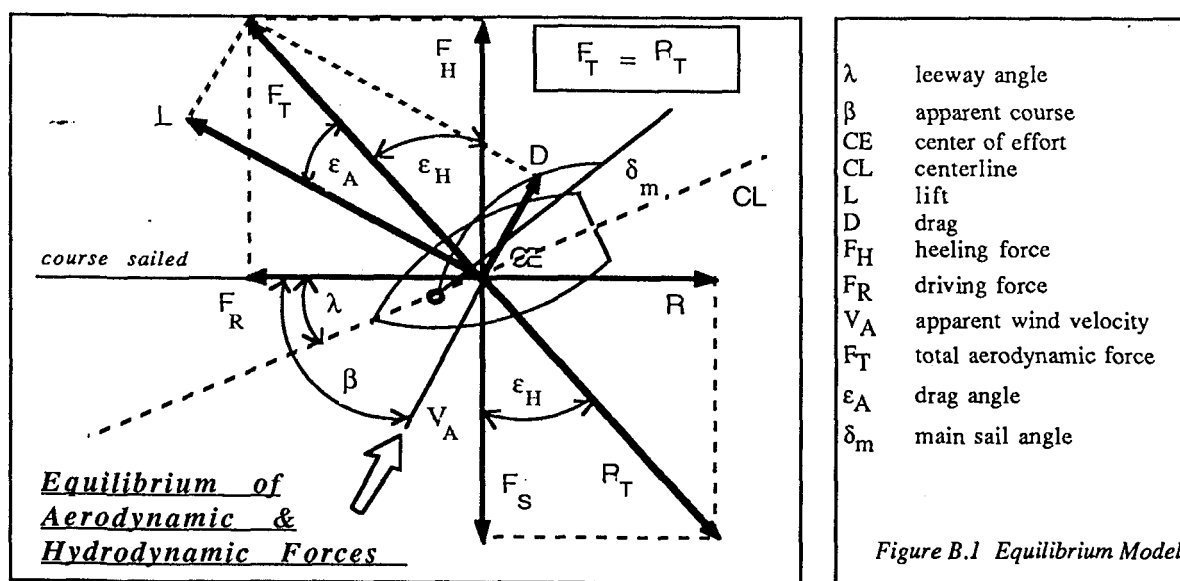
The following is a list of sailing terms used in this document. This list is taken from [Grubb79].

Abeam	At right angles to the side of the boat.
Actual Wind	The direction and force of wind felt on a boat that is not moving (also true wind).
Aft	Toward or near the stern.
Apparent Wind	The direction and force of the wind felt on a moving boat.
Bear	To move in a specified direction: 1) bear up means to turn to windward; 2) bear off or bear away means to turn to leeward or away from the wind; and 3) bear down means to approach another boat from windward.
Beat	To sail to windward.
Bow	The front of the boat.
Close Hauled	Sailing as close to the wind as possible (i.e. as far as possible upwind or to weather).
Come About	To tack by turning the bow of the boat across the wind.
Course	The direction sailed. Also the area and track of a race.
Down wind	Sailing away from the direction the wind is blowing.
Heading	The course of direction in which a boat is travelling.
Head Up	To steer the boat toward the wind. Also head off which is to steer away from the wind.
Jibe	To change course by steering the stern of the boat through the eye of the wind.
Lee	The side of a boat facing away from the wind.
Luff	To head the bow of a boat into the wind, causing the leading edge of the sail to shake.
Mark	Designates the turning point at the end of each leg (segment) of a race course.
Overlap	Refers to two boats in close proximity on the same course line. When the bow of the rearward boat extends past the stern of the forward boat an overlap occurs, which can be a tactical asset.
Points of Sail	The three general angles in which a sailboat can move forward: to weather or upwind, downwind and reaching.
Port	The left side of a boat as you face forward toward the bow.
Port Tack	Sailing with the wind blowing from the left side (port), and the mainsail on the right (starboard) side.
Reaching	Sailing with the wind coming essentially abeam.
Sheet	A line or rope which can be tightened or slackened to control the sail.
Starboard	The right side of the boat as you face forward toward the bow.
Starboard Tack	Sailing with the wind blowing from the right side (starboard), and the mainsail is on the left (port) side of the boat.
Stern	The rear of the boat.
Tack	To come about, change course by steering the bow of the boat through the eye of the wind (i.e. across the wind).

Trapeze	A wire hanging from the mast which allows, using a harness, the crew and/or skipper to stand on the windward rail to improve boat trim.
Tiller	A handle connected to the rudders which controls their angle and thus steers the boat.
Traveller	Horizontal track mounted on the boat to control the trim of the sail.
Trim	To set the sails at their most efficient shape and juxtaposition to the wind (necessary for achieving maximum performance).
Upwind	Sailing toward the wind.

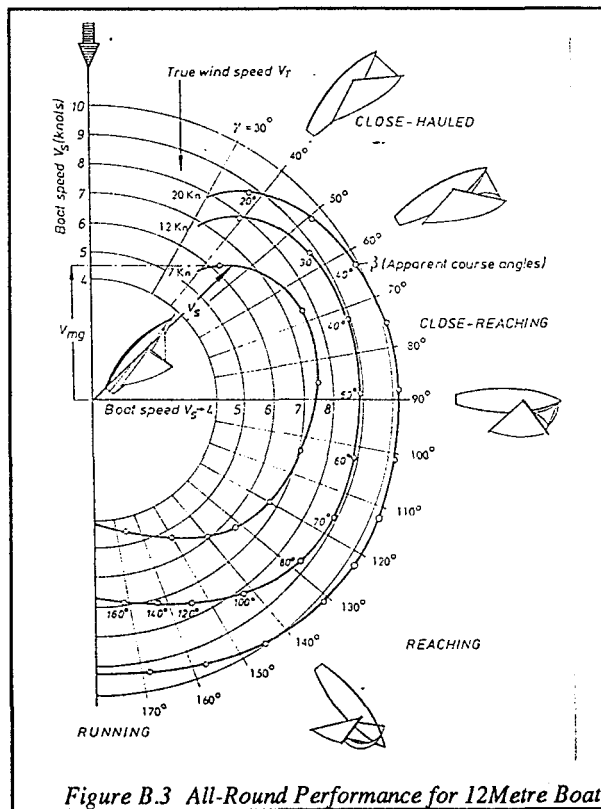
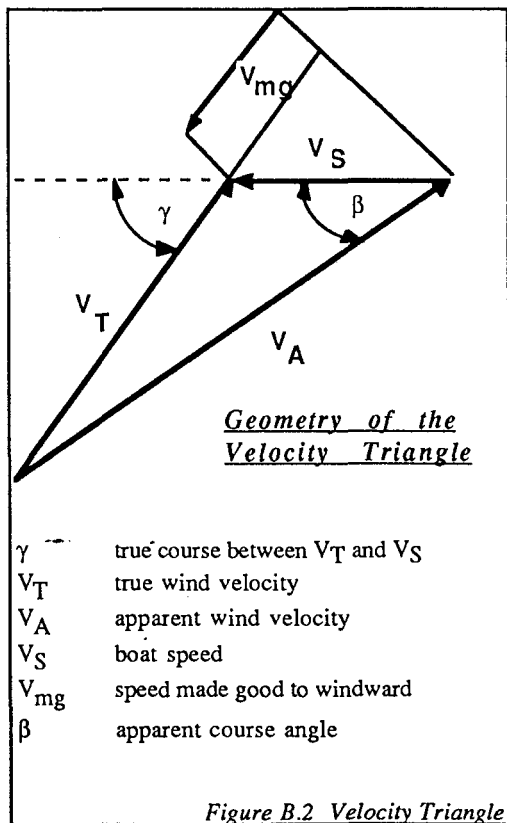
B.2 PREDICTING SAILBOAT PERFORMANCE

Predicting the performance of a sailboat requires very complex computations. In fact, there is no exact method of *computing* a sailboat's overall performance [Marchaj88]. Boat performance, in terms of the interaction between aerodynamic and hydrodynamic forces and moments, is modelled mainly from experimental measurements in wind tunnels and drag tanks. In this section, the equilibrium between opposing systems of forces on a sailboat is depicted graphically. This equilibrium model (Figure B.1) and experimental data are used to find values for the velocity triangle for sailing in Figure B.2. These values are then plotted as polar performance diagrams. The implementation of NSAIL extracts the information from a polar performance diagram for 12 metre sailing yachts (Figure B.3), and applies it to the calculation of boat speed given the boat heading and wind direction. The points of the performance diagram are fit to a curve for interpolation of intermediary points and wind velocities. Estimation of the impact of the sail angle on boat performance is based on a similar approach using collected data correlating sail angle to boat speed.



A sailboat moves on the boundary between air and water, being partly immersed in each. Its motion is the result of a complex system of forces acting on the sailboat, including gravitational, aerodynamic and hydrodynamic forces. This system of forces, in order to attain equilibrium²⁸, must satisfy the following criteria: 1) the sum of the vertical forces must equal zero; 2) the sum of the horizontal forces must equal zero; and 3) the sum of the moments, responsible for the turning effects of the forces, must also equal zero.

The results of experiments in wind tunnels, aerodynamic theory and experimentation with real sailboats reveal the dependence of the total aerodynamic force (F_T in Figure B.1) on various factors including the following: 1) the dynamic pressure of the apparent wind; 2) the total sail area; 3) the angle of incidence of the sail(s); 4) the shape of the sail(s) as defined by its cut and camber; and 5) the quality of the sail material (e.g. stiffness, weight, porosity, smoothness, etc.). The impact of all these factors on boat performance have been studied for yacht design, and supply a large volume of data for simulation.



²⁸One of Newton's Four Laws of Motion, the Principle of Inertia, states: Every body remains at rest or in uniform motion in a straight line when acted on by a system of forces whose resultant force is zero.

REFERENCES.

- Abel85 Robert Abel and Associates. *SIGGRAPH Video Review, Issue 20: The Making of Brilliance*. ACM, New York, 1985.
- Agha86 G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. The MIT Press, Cambridge, MA, 1986.
- Alias Alias Research Inc. 500-110 Richmond St. East, Toronto, Ont., Canada.
- Amkraut85 S. Amkraut, M. Girard, and G. Karl. *Motion Studies for Eurythmy*. SIGGRAPH Video Review, Issue 21, 1985.
- Badler82 N. I. Badler. *Modelling the Human Body for Animation*. IEEE Computer Graphics and Applications 2(9), November, 1982, pp. 6-7.
- Badler86 N. I. Badler. *Animating Human Figures: Perspectives and Directions*. Proc. of Graphics Interface 1986, pp. 115-120.
- Badler87 N. I. Badler, K. H. Manoochehri and G. Walters. *Articulated Figure Positioning by Multiple Constraints*. IEEE Computer Graphics and Applications, Vol. 7, No. 6, 1987, pp. 39-51.
- Badler89 N. I. Badler. *Artificial Intelligence, Natural Language, and Simulation for Human Animation*. State-of-the-Art in Computer Animation, D. Thalmann (ed.), Springer-Verlag, pp. 19-31, 1989.
- Badler90 N. I. Badler, B. Webber, J. Kalita, J. Esakov. *Animation from Instructions. Making Them Move: Mechanics, Control, and Animation of Articulated Figures*. N. Badler, B. Barsky, D. Zeltzer (eds.), Morgan Kaufmann Publishers, San Mateo, California, pp. 51-93, 1990.
- Badler92 N. I. Badler. *The Use of Natural Language in Human Animation*. SIGGRAPH Course Notes (#17): Advanced Techniques in Human Modelling, Animation and Rendering, July, 1992, pp. 133-178.
- Barzel87 R. Barzel and A. H. Barr. *A Modeling System Based on Dynamic Constraints*. Proc. ACM SIGGRAPH 88, Computer Graphics 22(4), 1988, pp. 179-188.
- Boden77 M. A. Boden. *Artificial Intelligence and Natural Man*. Basic Books, NY, 1977.
- Brachman85 R. J. Brachman and H. Levesque. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Gatos, CA, 1985.
- Braitenberg84 V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, Cambridge, MA, 1984.
- Brett89 C. Brett, S. Pieper and D. Zeltzer. *Putting it all together: An Integrated Package for Viewing and Editing 3D Microworlds*. Implementing and Interacting with Real-time Microworlds, D. Zeltzer (ed.), SIGGRAPH 1989 Course Notes, Boston, Mass, pp. 5-1 to 5-15.

References

- Brooks83 R. A. Brooks. *Solving the Find Path Problem by Good Representation of Free Space*. IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, No. 3, 1983, pp. 190-196.
- Brown89 D. C. Brown, B. Chandrasekaran. *Design Problem Solving: Knowledge Structures and Control Strategies*. Morgan Kaufmann Publishers, San Mateo, California, 1989.
- Bruderlin88 A. Bruderlin. *Goal-Directed, Dynamic Animation of Bipedal Locomotion*. Master's Thesis. School of Computing Science, Simon Fraser University, 1988. (Also technical report no. CMPT-TR-88-10.)
- Bruderlin89 A. Bruderlin and T. Calvert. *Goal-Directed, Dynamic Animation of Human Walking*. Proc. of ACM SIGGRAPH 89, Computer Graphics, 23(3), 1989, pp. 233-242.
- Bruderlin90 A. Bruderlin. *Dynamics + Kinematics = Controllable Realism*. Human Figure Animation: Approaches and Applications, ACM SIGGRAPH 90 Notes for Course No. 8, 1990, pp. 136-144.
- Bruderlin93 A. Bruderlin and T. Calvert. *Interactive Animation of Personalized Human Locomotion*. Proc. Graphics Interface 1993 (to be published).
- Bruynooghe84 M. Bruynooghe and L. M. Pereira. *Deduction Revision by Intelligent Backtracking*. Implementations of Prolog, J. A. Campbell (ed.), Ellis Horwood Publishers, Chichester, England, 1984.
- Burtnyk71 N. Burtnyk and M. Wein. *Computer Generated Key-Frame Animation*. Journal of SMPTE 80, 1971, pp. 149-153.
- Calvert80 T. W. Calvert, J. Chapman and A. Patla. *The Integration of Subjective and Objective Data in the Animation of Human Movement*. Proc. of ACM SIGGRAPH 80, Computer Graphics 14, 1980, pp. 198-203.
- Calvert82 T. W. Calvert, J. Chapman and A. Patla. *Aspects of the Kinematic Simulation of Human Movement*. IEEE Computer Graphics and Applications 2, pp. 41-50.
- Calvert88 T. W. Calvert. *The Challenge of Human Figure Animation*. Proceedings of Graphics Interface 1988, pp. 203-210.
- Calvert93 T. W. Calvert, A. Bruderlin, S. Mah, T. Schiphorst and C. Welman. *The Evolution of a User Interface for Dance Choreography*. Proc. InterCHI 1993 (to be published).
- Catmull78 E. Catmull. *The Problems of Computer-Assisted Animation*. Proc. ACM SIGGRAPH 78, Computer Graphics 12, 1978, pp. 348-353.
- Cercone86 N. Cercone and G. McCalla. *Accessing Knowledge Through Natural Language*. In the 25th Anniversary Issue of Advances in Computers. M. Yovits (ed.). Academic Press, NY, 1986, pp. 1-99.
- Cercone87 N. Cercone. *Knowledge Representation: An Overview*. Indian Journal of Technology, 25, December 1987, pp. 521-543.
- Clocks81 W. F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin Heidelberg, 1981.

References

- Colgate73 S. Colgate. *Colgate's Basic Sailing Theory*. Van Nostrand Reinhold Incorporated, New York, 1973.
- Colmerauer73 A. Colmerauer, H. Kanoui, H. Pasero and P. Roussel. *Un systeme de communication homm-machine en Francais*. Aix-Marseille University, France, 1973.
- Colmerauer90 A. Colmerauer. *An Introduction to Prolog III*. Communications of the ACM, 33(7), 1990, pp. 69-90.
- Drewery86 K. Drewery and J. Tsotsos. *Goal Directed Animation Using English Motion Commands*. Proc. of Graphics Interface 1986, pp. 131-135.
- Esakov91 J. Esakov and N. I. Badler. *An Architecture for High-Level Task Animation Control*. Knowledge-Based Simulation: Methodology and Application. Springer-Verlag, Berlin Heidelberg, 1991, pp. 162-199.
- Fikes85 R. Fikes and T. Kehler, *The Role of Frame-Based Representation in Reasoning*. CACM, 1985, pp. 904-920.
- Forsey88 D. Forsey and J. Wilhelms. *Techniques for Interactive Manipulation of Articulated Bodies Using Dynamics Analysis*. Proc. Graphics Interface 1988, pp. 8-15.
- Freuder78 E. C. Freuder. *Synthesizing Constraint Expressions*. Communications of the ACM, 21(11), 1978, pp. 958-966.
- Genesereth87 M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1987.
- Gill81 P. Gill, W. Murray and M. Wright. *Practical Optimization*. Academic Press, New York, N.Y., 1981.
- Ginsberg83 C. Ginsberg and D. Maxwell. *Graphical Marionette*. Proceedings of ACM SIGGRAPH/SIGART Workshop on Motion, 1983, pp. 172-179.
- Girard85 M. Girard and A. A. Maciejewski. *Computational Modeling for the Computer Animation of Legged Figures*. Proceedings of SIGGRAPH 85, 19, pp. 263-270, July, 1985.
- Gomez85 J.E. Gomez. *Twixt: 3D Animation System*. Computer & Graphics 9(3), 1985, pp. 291-298.
- Gould89 R. L. Gould. *Making 3-D Computer Character Animation: A Great Future of Unlimited Possibility or Just Tedious?* SIGGRAPH 89 Course Notes: 3D Character Animation by Computer, 1989, pp. 31-60.
- Grubb79 J. Grubb. *Hobie Cat Sailing*. Airguide Publications, Long Beach, California, 1982.
- Hadley85 R. Hadley. *SHADOW: A Natural Language Query Analyser*. MSc Thesis, Computing Science, Simon Fraser University, 1983. Also in *Computers & Mathematics*. Pergamon Press, Oxford, 1985.

References

- Hanrahan85 P. Hanrahan and D. Sturman. *Interactive Animation of Parametric Models*. The Visual Computer: International Journal of Computer Graphics 1(4), December 1985, pp. 260-266.
- Harmon88 P. Harmon, R. Maus and R. Morrissey. *Expert Systems Tools and Applications*. John Wiley & Sons Inc., Toronto, 1988.
- Haumann88 D.R. Haumann and R.E. Parent. *The behavioral Test-bed: Obtainin Complex Behavior from Simple Rules*. The Visual Computer, 4(6), 1988, pp. 332-347.
- Havens90 W. Havens. *Echidna Constraint Reasoning System: Programming Specifications*. Proc. of Computational Intelligence 90, Milano, Italy, September 1990, pp. 24-28.
- Havens91 W. Havens. *Intelligent Backtracking in the Echidna CLP Reasoning System*. TR No. CSS-IS-TR-91-07, Simon Fraser University, 1991. Also in The International Journal of Expert Systems: Research and Applications (in press).
- Havens92 W. Havens, S. Sidebottom, G. Sidebottom, J. Jones and R. Ovens. *ECHIDNA: A Constraint Logic Programming Shell*. TR No. CSS-IS-TR-92, Simon Fraser University, 1992. Also in Proc. of the 1992 Pacific Rim International Conference on Artificial Intelligence (Seoul, Korea).
- Hayes81 P. Hayes. *The Logic of Frames*. In Readings in Artificial Intelligence. B. Webber and N. Nilsson (eds.). Tioga Publishing, Palo Alto, CA., 1981, pp. 451-458.
- Hewitt72 C. Hewitt. *Description and Theoretical Analysis of PLANNER*. PhD Thesis, MIT, Cambridge, MA, 1972.
- Issacs87 P. M. Issacs and M. F. Cohen. *Controlling Dynamic Simulation with Kinematic Constraints*. Proc. of ACM SIGGRAPH 87, Computer Graphics 21(4), July, 1987, pp. 215-224.
- Jaffar87 J. Jaffar and S. Michaylov. *Methodology and Implementation of a CLP system*. Proceedings of the Fourth International Conference on Logic Programming, Melbourne, Aust., 1987.
- Karp90 P. Karp and S. Feiner. *Issues in the Automated Generation of Animated Presentations*. Proc. Graphics Interface 1990, pp. 39-48.
- Karp93 P. Karp and S. Feiner. *Automated Presentation Planning of Animation Using Task Decomposition with Heuristic Reasoning*. Proc. Graphics Interface 1993 (to be published).
- Kass92 M. Kass. *CONDOR: Constraint-Based Dataflow*. Proceedings of SIGGRAPH 1992, Computer Graphics 26(2), pp. 321-330.
- Kochanek84 D. Kochanek and R. Bartels. *Interpolating Splines with Local Tension, Continuity and Bias Tension*. Proc. ACM SIGGRAPH 1984, pp. 33-41.
- Korein82 J. U. Korein and N. I. Badler. *Techniques for Generating the Goal-Directed Motion of Articulated Structures*. IEEE Computer Graphics and Applications 2(9), November 1982, pp. 71-81.

References

- Kowalski79 R. Kowalski. *Predicate Logic as Programming Language*. Proc. of the IFIP Congress 74, North Holland (ed.), 1979, pp. 569-574.
- Kuipers86 B. J. Kuipers. *Qualiative Simulation*. Artificial Intelligence, 29(3), September, 1986, pp. 280-338.
- Lansky88 A. Lansky. *Localized Event-Based Reasoning For Multi-Agent Domains*. Computational Intelligence: Special Issue on Planning, 4(4), 1983, pp. 319-340.
- Langton89 C. G. Langton (ed.). *Artificial Life*. Santa Fe Institute Studies in the Sciences of Complexity, VI, Reading, MA, Addison-Wesley, 1989.
- Lasseter87 J. Lasseter. *Principles of Traditional Animation Applied to 3D Computer Animation*. Proc. ACM SIGGRAPH 87, Computer Graphics 21(4), 1987, pp. 35-44.
- Lee89 P. Lee, C. Phillips, E. Otani and N. I. Badler. *The JACK Interactive Human Model*. Concurrent Engineering of Mechanical Systems, Vol. 1, First Annual Symposium on Mechanical Design in a Concurrent Engineering Environment, University of Iowa, Iowa City, IA, October 1989, pp. 179-198.
- Lethebridge89 T.C. Lethebridge and C. Ware. *A Simple Heuristically-Based Method for Expressive Stimulus-Response Animation*. Computers and Graphics, 13(3), 1989, pp. 297-303.
- Levy92a S. Levy. *Artificial Life: A New Quest for Creation*. Pantheon, New York, 1992
- Levy92b S. Levy. *Artificial Life*. Siggraph 92 Panel Discussion, Computer Graphics, 26(2), July, 1992, pp. 406-407.
- Lloyd84 J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, NY, 1984.
- Lorzano82 T. Lozano-Perez. *Automatic Planning of Manipulator Transfer Movements*. Technical Report AI Memo 606, MIT, 1982.
- McCalla83 G. McCalla and N. Cercone. *Approaches to Knowledge Representation*. Computer 16(10), 1983, pp. 12-16.
- McCarthy69 J. McCarthy and P. J. Hayes. *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. In Machine Intelligence 4, B. Meltzer and D. Michie (eds.), Edinburgh University Press, Edinburgh, 1969.
- McCarthy80 J. McCarthy. *Circumscription: A Non-Monotonic Inference Rule*. Artificial Intelligence, 13, 1980, pp. 27-40.
- Mackworth77 A. Mackworth. *Consistency in Networks of Relations*. Artificial Intelligence, 8, 1977, pp. 99-118.
- Magenat86 N. Magnenat-Thalmann and D. Thalmann. *EXPERTMIRA: An A.I. Language for Image Synthesis and Animation*. Technical Report, MIRALab, Universite de Montreal, 1986.
- Maiocchi90 R. Maiocchi and B. Pernici. *Directing an Animated Scene with Autonomous Actors*. Computer Animation 90, N. Magnenat-Thalmann and D. Thalmann (eds.), 1990, pp. 41-60.

References

- Marchaj64 C. A. Marchaj. *Sailing Theory and Practice*. Dodd, Mead & Co., New York, 1964.
- Marchaj88 C. A. Marchaj. *Aero-Hydrodynamics of Sailing (Revised and Expanded Edition)*. International Marine Publishing, Camden, Maine, 1988.
- Minsky75 M. Minsky. *A Framework for Representing Knowledge*. The Psychology of Computer Vision, P. Winston (Ed.), McGraw-Hill Publishers, New York, 1975.
- Morawetz89 C. L. Morawetz. *A High-Level Approach to the Animation of Human Secondary Movement*. Master's Thesis. School of Computing Science, Simon Fraser University, 1989.
- Morawetz90 C. L. Morawetz. *Goal-Directed Human Animation of Multiple Movements*. Proc. Graphics Interface 90, 1990, pp. 60-67.
- Muehle87 E. Muehle. *FROBS: A Merger of Two Knowledge Representation Paradigms*. Master's Thesis. University of Utah, 1987.
- Naish85 L. Naish. *Automatic Control for Logic Programs*. Journal of Logic Programming, 2(3), October 1985, pp.167-184.
- Newell76 A. Newell and H. A. Simon. *Computer Science as Empirical Inquiry: Symbols and Search*. CACM 19(3), March, 1976.
- Newell80 A. Newell. *Reasoning, Problem Solving and Decision Process: The Problem Space as a Fundamental Category*. Attention and Performance VIII. L. Erlbaum, 1980, pp. 693-718.
- Nielsen91 N. R. Nielsen. *Application of Artificial Intelligence Techniques to Simulation*. Knowledge Based Simulation: Methodology and Application. P. A. Fishwick and R. B. Modjeski (Eds.). Springer-Verlag, New York, 1991, pp. 2-19.
- Nilsson71 N. Nilsson. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.
- ODonnel81 T. J. O'Donnell and A. J. Olson. *GRAMPS- A Graphical Language Interpreter for Real-Time, Interactive, Three-Dimensional Picture Editing and Animation*. Proc. ACM SIGGRAPH 81, Computer Graphics, August, 1981, pp. 133-142.
- Parke82 F. I. Parke. *Parameterized Models for Facial Animation*. IEEE Computer Graphics and Applications 2, 1982, pp. 61-68.
- Phillips88 C. Phillips and N. I. Badler. *Jack: A Toolkit for Manipulating Articulated Figures*, Proc. ACM SIGGRAPH Symposium on User Interface Software, Banff, Canada, 1988.
- Prusinkiewicz90 P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, NY, 1990.
- Reeves83 W. Reeves. *Particle Systems: A Technique for Modelling a Class of Fuzzy Objects*. Proc. ACM SIGGRAPH 83, Computer Graphics 17(3), July 1983, pp. 359-376.

References

- Renault90 O. Renault, N. Magnenat-Thalmann, D. Thalmann. *A Vision-Based Approach to Behavioural Animation*. The Journal of Visualization and Computer Animation, 1(18), 1990, pp. 18-21.
- Reynolds82 C. W. Reynolds. *Computer Animation with Scripts and Actors*. Proc. of ACM SIGGRAPH 82, Computer Graphics 16(3), 1982, pp. 289-296.
- Reynolds87 C. W. Reynolds. *Flocks, Herds and Schools: A Distributed Behaviour Model*. Proc. ACM SIGGRAPH 87, Computer Graphics 21(4), 1987, pp. 25-34.
- Rich83 E. Rich. *Artificial Intelligence*. McGraw-Hill Book Company, New York, 1983.
- Ridsdale86 G. Ridsdale and T. Calvert. *The Interactive Specification of Human Animation*. Proc. Graphics Interface 1986, pp. 121-130.
- Ridsdale87 G. Ridsdale. *The Director's Apprentice: Animating Figures in a Constrained Environment*. PhD Thesis, School of Computing Science, Simon Fraser University. TR No. CMPT-TR-87-6, 1987.
- Ridsdale90 G. Ridsdale and T. Calvert. *Animating Microworlds from Scripts and Relational Constraints*. Computer Animation 90, N. Magenat-Thalmann and D. Thalmann (eds.), 1990, pp. 107-117.
- Rummelhart76 D. E. Rummelhart and A. Ortony. *The Representation of Knowledge in Memory*. TR, Center for Human Information Processing, Department of Psychology, University of California at San Diego, LaJolla, CA. 1976.
- Schaffer93 D. Schaffer et al. *Comparing Fisheye and Full-Zoom Techniques for Navigation of Hierarchically Clustered Networks*. Proc. Graphics Interface 1993 (to be published).
- Schefflen72 A. E. Schefflen. *Body Language and the Social Order*. Prentice-Hall Inc., Eaglewood Cliffs, NJ, 1972.
- Schiphorst90 T. Schiphorst, T. Calvert, C. Lee, C. Welman and S. Gaudet. *Tools for Interaction with the Creative Process of Composition*. Proc. CHI, 1990, pp. 167-174.
- Sidebottom92a S. Sidebottom. *Echidna Constraint Reasoning System Tutorial: Heuristics*. Tutorial Notes for ECHIDNA, 1992.
- Sidebottom92b S. Sidebottom. *Echidna Constraint Reasoning System Programming Manual (version 1.0)*. Expert Systems Laboratory, Centre for Systems Science, Simon Fraser University, 1992.
- Simon69 H.A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, MA, 1969.
- Sims87 K. Sims. *Locomotion of Jointed Figures Over Complex Terrain*. SM Thesis, MIT, April 1987.
- Sims91 K. Sims. *Artificial Evolution for Computer Graphics*. Computer Graphics 25(4), July, 1991, pp. 319-328.
- Sterling86 L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, Cambridge, Mass., 1986.

References

- Sturman87 D. Sturman. *A Discussion on the Development of Motion Control Systems*. Computer Animation: 3-D Motion Specification and Control, SIGGRAPH 87 Notes for Course No. 10, 1987, pp. 3-16.
- Sutherland63 I. E. Sutherland. *SKETCHPAD: A Man-Machine Graphical Communication System*. Cambridge, MA, MIT Lincoln Labs, 1963.
- Thalman86 D. Thalman, N. Magnenat-Thalman. *AI in Three-Dimensional Computer Animation*. Computer Graphics Forum 5, 1986, pp. 341-348.
- Thalman88 D. Thalman, N. Magnenat-Thalman, B. Wyvill, D. Zeltzer. *Synthetic Actors: The Impact of Artificial Intelligence and Robotics on Animation*, Siggraph 88 Course Notes #4, Boston, 1988.
- Thalman92 D. Thalman, N.I. Badler, N. Magnenat-Thalman, and D. Terzopoulos. *Advanced Techniques in Human Modelling, Animation and Rendering*, Siggraph 92 Course Notes #17, Chicago, 1992.
- vanBaerle86 S. van Baerle. *Computer Animation: Combining Computer Graphics with Traditional Animation*. Graphics Interface 86: Tutorial on Computer Animation, 1986.
- VanHentenryck89 P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, Cambridge, MA, 1989.
- Vertigo Vertigo Technology Inc. 1010-1030 West Georgia Street, Vancouver, B.C., Canada.
- Waltz72 D. Waltz. *Generating Semantic Descriptions from Drawings of Scenes with Shadows*. TR-AI271, MIT, MA, November, 1972.
- Webber92 B. L. Webber. *Animation from NL Instructions*. Advanced Techniques in Human Modelling, Animation and Rendering, ACM SIGGRAPH 92 Notes for Course No. 17, 1992, pp. 146-166.
- Wilhelms85 J. Wilhelms and B. Barsky., *Using Dynamic Analysis to Animate Articulated Bodies Such as Humans and Robots*. Proc. Graphics Interface 85, May 1985, pp. 197-204.
- Wilhelms87 J. Wilhelms. *Toward Automatic Motion Control*. IEEE Computer Graphics and Applications, Vol. 7, No. 4, April 1987, pp. 11-22.
- Wilhelms89 J. Wilhelms and R. Skinner. *An Interactive Approach to Behavioural Control*. Proc. Graphics Interface 1989, pp. 1-8.
- Wilhelms90 J. Wilhelms. *Behavioural Animation Using An Interactive Network*. Proc. Computer Animation 1990, pp. 95-105.
- Wilkins84 D. Wilkins. *Domain-Independent Planning: Representation and Plan Generation*. Artificial Intelligence, 22, 1984, pp. 269-301.
- Wilkins88 D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc., San Mateo, California, 1988.
- Winograd72 T. Winograd. *Understanding Natural Language*. Academic Press, New York, 1972.

References

- Witkin88 A. Witkin and M. Kass. *Spacetime Constraints*. Computer Graphics, 22, No. 4, August 1988, pp. 159-168.
- Wolfram86 D. A. Wolfram. *Intractable Unifiability Problems and Backtracking*. Third International Conference on Logic Programming, London, U.K., July, 1986.
- Yaeger82 L. Yaeger. *Interactive Evolution of Dynamical Systems*. Proceedings of the First European Conference on Artificial Life, MIT Press, Cambridge, 1982.
- Zeltzer82 D. Zeltzer. *Motor Control Techniques for Figure Animation*. IEEE Computer Graphics and Applications, 2(9), November 1982, pp. 53-59.
- Zeltzer83 D. Zeltzer. *Knowledge-Based Animation*. Proceedings of SIGGRAPH/SIGART Workshop on Motion. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1983, pp. 187-192.
- Zeltzer85 D. Zeltzer. *Towards an Integrated View of 3D Computer Animation*. The Visual Computer, 1(4), 1985, pp. 249-259.