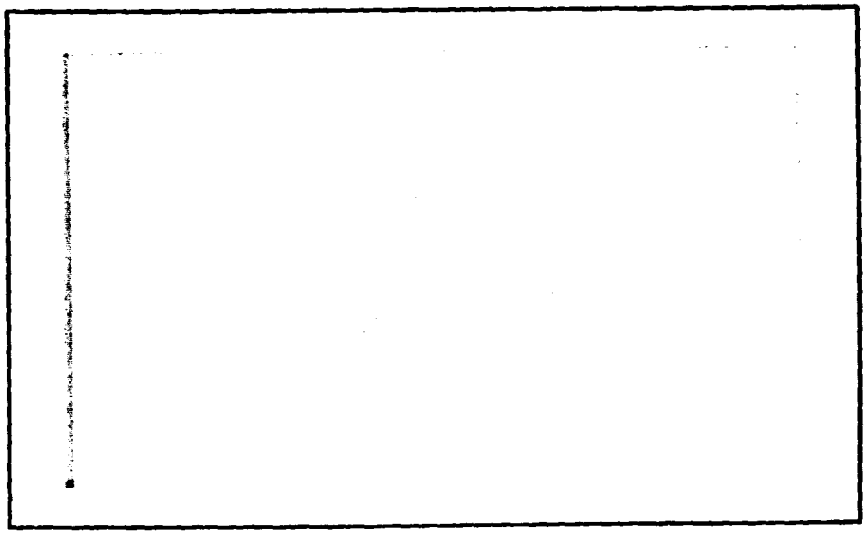
CR-171899

C.1



(NASA-CR-171899) ZERO-GRAVITY MOVEMENT STUDIES (Pennsylvania Univ.) 113 p
HC A06/MF A01 CSCL 12B

N86-10897

Unclas
G3/66 27494

UNIVERSITY of PENNSYLVANIA

The Moore School of Electrical Engineering

PHILADELPHIA, PENNSYLVANIA 19104

Zero-Gravity Movement Studies

May 31, 1985

NASA Contract No. NAS9-16634

Norman I. Badler,
Principal Investigator

Paul Fishwick
Nina Taft
Mukul Agrawala

Department of Computer and Information Science
Moore School D2
University of Pennsylvania
Philadelphia, PA 19104

Table of Contents

List of Figures

List of Tables

1. Introduction

The use of computer graphics to simulate the movement of articulated animals and mechanisms has a number of uses ranging over many fields. Human motion simulation systems can be useful in education, medicine, anatomy, physiology, and dance. In biomechanics, computer displays help to understand and analyze performance. Simulations can be used to help understand the effect of external or internal forces. Similarly, zero-gravity simulation systems should provide a means of designing and exploring the capabilities of hypothetical zero-gravity situations before actually carrying out such actions. The advantage of using a simulation of the motion is that one can experiment with variations of a maneuver before attempting to teach it to an individual.

We can divide the zero-gravity motion simulation problem into two broad areas: human movement and behavior in zero-gravity, and simulation of articulated mechanisms. We will examine each in turn.

In the absence of external forces the linear momentum of a body is conserved. Similarly in the absence of torques the angular momentum of a body is conserved. We can examine the motion of a diver (which is a good example of effective free-fall) and then extended the analogy to that of an astronaut [34, 20]. The physics of rotational motion in free-fall examines motion relative to the center of mass point. The center of mass is defined as a mathematical point whose position is determined by the distribution of mass within the body. Rotations of a body in free-fall are about the center of mass.

The basic question is the following: how can a diver in mid-air suddenly twist and somersault without violating the law of the conservation of angular momentum? The somersault is defined as the basic rotation about the axis through his waist and the twist is the rotation about the longitudinal axis running from his head to his toe. Angular momentum is the product of the angular velocity and the moment of inertia $H=I\omega$. Both H and I are vectors. The moment of inertia of a rigid body about an axis is the body's tendency to resist changes in angular velocity about that axis. It is possible for the diver to change his rate of spin (angular velocity) if he decreases his moment of inertia so the angular moment remains the same. For example, if he brings his arms and legs in closer to the longitudinal axis while spinning he decreases his moment of inertia and increases his spin rate. It is also possible for a diver to change both his somersaulting angular momentum and his twisting momentum as long as their sum, total angular momentum, remains constant in magnitude and direction. This situation closely parallels that of the

astronaut in that a man working in space in a weightless environment must be able to control his body orientation. He can start in a motionless position and with a few simple movements reorient himself in any direction. The underlying physics is the same for an astronaut as it is for a diver since both men are moving in the absence of torques.

Ramey and Yang have published a detailed procedure to describe human motion in three-dimensional space [30]. The motions under study are those occuring in free-fall, however the procedure developed includes the effects of external forces at the initial stage. Ramey and Yang use a nine body-segment model. The equation of motion is developed using the principle of conservation of angular momentum referred to the mass center of the body.

Most existing computer graphics body modeling systems are kinematic not dynamic [23, 5, 7, 14, 18, 22], in other words, they only consider joint positions and angles. Kinetic systems use motion variables as position, velocity, and accelerations and have been mostly used in crash simulation [31, 19]. A dynamic system will also consider forces and torques [35]. Since dynamic systems are more realistic than kinematic ones, a zero-gravity simulator must be a dynamic system. At least one recent attempt to produce a dynamics simulator for articulated figures has been reported by Wilhelms [35]. Unfortunately her reliance on purely dynamic simulation leads to difficulty in controlling jointed motions, most notably at the point where free motion contacts ground or another obstacle. Apparently both kinematics and dynamics are required for effective control. Girard and Maciejewski have successfully merged kinematic control of multiple leg movements with overall dynamic control of the "body" along an arbitrary motion path [21]. Their Jacobian and pseudo-inverse techniques are different from those used for kinematic reach in TEMPUS, but the results are very efficient and promising for future simulation and control systems.

2. Requirements For Free-fall Jointed Body Dynamics

Unrestrained motion in zero gravity must be examined from three perspectives: the dynamics of pushing and grasping, the transition between grasping or pushing points (including translation in free-fall), and the path planning problem (including collision detection and avoidance). Our approach to these problems consists of a study of the physics involved, then a decomposition of the problem into components which offer the best cost-effective solution involving the computer and an operator. This interaction is much more likely to achieve satisfactory results than either could do alone. For example,

the state of the art in robot motion path planning and collision avoidance is not yet capable of efficiently handling the full complexity of a multiply-jointed body in an arbitrary workplace [lozano79, brooks82], though the literature provides us with useful heuristics.

In one scenario, the operator would specify an initial and final body position and request TEMPUS to specify a feasible path of movement between the two positions. The path of movement is then computed and may be graphically animated. The 'feasibility' requirement dictates the following constraints on the movement:

1. It must be kinematically plausible. There should be no interpenetration of the moving person and solid objects, and joint limits must not be violated.

2. It must be dynamically plausible. The equations of motion of the moving body, both in free-fall and when interacting with external objects, must be satisfied.

These two constraints imply that both geometry and forces are essential for an adequate simulation. The former is rather obvious, but the latter is needed since zero-gravity motion is controlled by the "non-standard" application of forces. Therefore forces resulting from movements, such as the resistance of external objects and limitations on an individual's ability to apply such forces, should be modeled. For example, if a person should push himself off a wall, or grasp a restraint to slow or stop his motion, then numerous verifications must be made. The person must be capable of exerting the forces necessary to achieve the resulting positions and velocities. The wall must be able to withstand the resulting force and be a surface which is safe to push or pull against. The capabilities provided for strength analyses [3] must be used extensively.

The OSDS operator must have facilities for defining which objects in the environment may be used for grasping and pushing and the forces they will accomodate. The user may also partially specify paths by allowing only a subset of these objects to be used or by explicitly requiring the path to contain intermediate body positions. The latter is especially important in order to refine a path found to contain undesirable collisions.

In order to study the dynamics of grasping and pushing we must examine rigid body motions under the influence of external forces. In the next section we look at some experiments in computing free body motions.

2.1. Rigid Body Dynamics

First we examine the dynamics of a rigid body in free-fall under the application of external impulse forces. Then we will look at articulated objects and determine how their motions change when the angle of articulation changes.

Two experiments were performed to study specific situations of the kinematic and dynamic motion of an object. For study purposes the programs embody certain restrictions that simplify the mathematics and yet do not trivialize the problem. More complex cases may be handled by the essential simulation underpinnings. In particular, the objects manipulated are simple geometric forms of uniform mass, the forces applied are considered to be impulses at an instant of time, and articulated motions are assumed to occur instantaneously. By choosing a small enough time interval for the simulation, the resulting dynamics closely approximate most of the significant motion effects.

The forces acting on a body are gravity, inertia, friction, centrifugal, and Coriolis. Each of these may be modeled as an impulse over a small enough time interval. Likewise, motion of the joints of the articulated object are assumed to occur at a "slow enough" rate so that we can ignore the effects of added torques for now. They, too, can be modeled as impulse forces if need be. While this method may not be the most efficient, it serves as a necessary first approximation. The robotics literature, though concerned with dynamics formulations, is most concerned with producing the "correct" joint torques to control a given motion of the end effector [10]. These torques are therefore related to the maximum torque that can be developed at each joint. For a human figure, this is a strength or joint limit problem. For locomotion, however, we are interested in motion of the body itself. It is significant to note that the robotics world can always assume a fixed base that can absorb virtually any forces or torques applied to it. Our figures in free-fall, however, are more likely to be moved by their environment rather than *vice versa*, since it is the environment that is more likely to be (relatively) massive and stabile.

The two demonstratable programs, called BOX and RODS, handle the two cases of rigid and articulated motion. The BOX program uses as its object a rectangular prism (solid box) and allows the user to enter a force (magnitude and direction) and the point of application of the force. The force input is actually an impulse, which means that the force is applied only for an instant in time; *i.e.* the force is not being applied constantly to accelerate the object. The resulting motion of the box involves both translational and

rotational motion. The program determines the translational and rotational parameters and transforms the object vertices accordingly. The program also allows the user to enter a small time period (ΔT) and it updates the values of these vertices every ΔT. The user can see the new position of the object after ΔT seconds, and its next new position after another ΔT seconds, *etc.*

The program RODS generates two end-connected rods of uniform, non-zero mass. The rods are assumed to be already moving with some initial angular velocity (arbitrarily set by the programmer). The angle between the two rods can be changed at any time. This angle change is what happens when a person applies internal forces at one of his joints to change one of his body segments in relation to the other. For example, he might bend his lower arm and move it closer to his upper arm. This change in angle between the two rods causes the center of mass of the system to change and alters the rotational motion. A new inertia matrix describing the system of objects is calculated and from that the resulting motion is determined. We will describe this further below.

The BOX program makes use of the routines *XYZMAT, getinpt, online, angular_momentum, PostMult* and the necessary CORE graphics routines. *XYZMat* and *PostMult* are existing TEMPUS routines. *XYZMat* determines the rotation matrix given the yaw, pitch and roll angles as input. *PostMult* applies a matrix to a vector and leaves the results in a new vector. This is the routine used to apply the rotation matrix to all the points (actually defined as vectors from the origin of the local coordinate system). *Getinpt* is a routine to get the minimal set of input needed from the user. *Online* is a function that determines whether a point lies on a given line. In this case, the point is the point of application and the line is the line defined by the center of mass point and the direction of the force. If the point of application lies on this line, then there is no rotational motion. This routine is useful because if we can know this information ahead of time all the calculations for rotational motion need not be carried out. The *angular_momentum* routine determines the angular momentum vector according to $H=M(r\times v)$ where $r\times v$ is the cross product of the point r (the point of application) and the velocity v. M is the total mass and H is the angular momentum. H is a vector because it has three components, one in the x-direction, one in the y-direction and one in the z-direction.

The RODS program uses the routines *NewCmSys, Rinertia,* and *Rod_len*, in addition to various TEMPUS routines. The *NewCmSys* routine takes as input the

coordinates of two rods, defined in the same coordinate system. The conversion of one rod's coordinates into coordinates of the other rod's local coordinate system is done in the main program. *NewCmSys* determines the coordinates of the center of mass of the system of objects (in this case the two rods) and then redefines the local coordinate system so that its origin is the current center of mass. Each time we change the angle between the two rods this routine must be called to determine the new center of mass. The *Rinertia* routine is the one that determines the inertia matrix of the system of rods about axes through the center of mass. First we must determine the inertia matrix for each rod and then we add the two matrices together. For an object composed of several simple bodies, we are allowed to add their respective inertia matrices as long as they are all defined relative to the same set of axes [6].

The general formula for the inertia of a rod defined along the x-axis is $Ixx=0$ and $Iyy=Izz=Mass \cdot Len^2$. For a rod along the y-axis, $Iyy=0$ and $Ixx=Izz=Mass \cdot Len^2$. (The formulas for the z-axis case follow similarly.) This product of mass and length squared would be the only component of the inertia if we were describing the inertia of each rod about axes through its own center of mass. But since we want to describe the inertia about axes through the center of mass of the system, we use the parallel axis theorem to add an additional factor to each of the elements of the inertia matrix. The parallel axis theorem says $I'=I+md^2$ where d is the distance between the current axis and the new parallel axis about which we wish to determine the inertia. The principal formula used to determine the rotational motion is $H=I\omega$, where H is the angular velocity vector, I is the inertia matrix, and ω is the angular velocity vector. We are given some initial angular velocity and determine the inertia matrix according to the input parameters describing the coordinates of the points of the rods and the relations between the various coordinate systems. We compute the angular momentum once and then it remains constant. This is because angular momentum is conserved in this situation. Hence, each time we change the angle between the two rods, we determine the new coordinates of the rods and determine the new inertia matrix. Since angular momentum is conserved, we can compute the new angular velocities: $\omega=[I^{-1}]H$. Initially, it was thought that the axis of rotation of the rigid object would be a useful parameter to determine, but actually the axis of rotation of the system changes every instant and for the moment does not appear to be of any use.

A test program prompts the user for the minimal set of input required for the *getinpt* routine, namely the force and the point of application. The program determines

the yaw, pitch and roll angles from this input. From these parameters and the object's definition, BOX determines the translational and rotational motion due to the impulse applied. First the velocity $V=F\cdot\Delta t/M$ is computed, then the translation $T=V\Delta t$. If there is rotational motion (determined by the function *online*) then the angular momentum, inertia, and angular velocity is determined. The three components of the angular velocity give the change in angle around each axis and these are the input parameters to *XYZMat* which produces the rotation matrix.

2.2. Inertia Routines

In general, the inertia matrix for a three dimensional object takes the following form:

$$\begin{matrix} Ixx & Ixy & Ixz & 0 \\ Iyx & Iyy & Iyz & 0 \\ Izx & Izy & Izz & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

The inertia matrix is dependent on the type of object (shape), its composition, and the coordinates in which the object is defined. If the object's local coordinate system is defined such that the object's center of mass is the origin of the coordinate system and the axes of the object (height, length and width) are defined along the coordinate system's axes, then all of the non-diagonal elements in the above matrix go to zero.

In general, the formulas for the elements in the matrix are based on integrating over the mass distribution. All of these formulas should be integrals with dM as the differentiable mass element. Hence the resulting values depend on the limits of integration, which depend in turn upon the shape of the object. The non-diagonal elements are called the products of inertia. The moments of inertia (diagonal elements) are:

$$Ixx=\int (y^2+z^2)dM$$

$$Iyy=\int (x^2+z^2)dM$$

$$Izz=\int (x^2+y^2)dM$$

The products of inertia are:

$$Ixy(=Iyx)=\int xy\,dM$$

$$Ixz(=Izx)=\int zx\,dM$$

$$Iyz(=Izy)=\int zy\,dM$$

The two inertia routines, *binertia* (BOX) and *rinertia* (RODS), contain the solutions for these particlar shapes. In the case of the box the non-diagonal elements are zero since the box is defined symmetrically in a local coordinate system This means that the center of mass of the box is the origin of the coordinate system and its coordinates along the axis are equivalent per axis (*i.e.* $|x|=|x|$, $|y|=|y|$, *etc.*). Let hx be the length, hy be the height, and hz be the width. Then the moments of inertia are (from the results of the integration):

$$Ixx = 1/12\ M(hy^2+hz^2)$$
$$Iyy = 1/12\ M(hx^2+hz^2)$$
$$Izz = 1/12\ M(hx^2+hy^2)$$

The results of the integration for the case of the rods depends on the axis upon which the rod is defined. If the rod is defined to lie along the x-axis, then the moments of inertia (where M is the mass and L is the length) are:

$$Ixx = 0$$
$$Iyy = Izz = 1/12\ M{\cdot}L^2$$

If the rod is defined along one of the other axes, the moments of inertia follow according to the same pattern.

If the rod is not defined along an axis, a simple transformation should be computed to transform the rod's coordinates into a coordinate system where it is defined along one of the axes. This specification of the rod is critical since the formulas in the inertia matrix ($I=(1/12)M{\cdot}L^2$) depend upon this configuration.

If we want to determine the inertia matrix in another coordinate system whose axes are parallel to the initial coordinate system, then an additional factor must be added to each element in the inertia matrix. The parallel axis theorem describes the additional factor [6]. The theorem says $I'=I+md^2$ where d is the distance between the two origins. Let (a,b,c) be the distance from the new coordinate system's origin to the old coordinate system's origin, with a, b and c defined in the new coordinate system. Let

x' denote the new coordinate and x denote the initial coordinate. The new coordinates are $x'=x+a$; $y'=y+b$; $z'=z+c$. To update the diagonal elements of the inertia matrix we add the extra factor as follows.

$$\begin{aligned} Iaa &= Ixx+M(b^2+c^2) \\ Ibb &= Iyy+M(a^2+c^2) \\ Icc &= Izz+M(a^2+b^2) \end{aligned}$$

The pattern is to add to the inertia around a given axis the product of the total mass M and a special quantity. This quantity is the sum of the squares of the two distances different from the axis which we are determining. In other words, for axis a (or new x), the special quantity is the sum of the squares of b and c; for the inertia about axis b, the special quantity is the sum of the squares of a and c; *etc.* The factor which must be added to the products of inertia follows a different pattern. The new products are:

$$\begin{aligned} Iab &= Ixy+Mab \\ Ibc &= Iyz+Mbc \\ Iac &= Ixz+Mac \end{aligned}$$

The pattern is to add the quantity which is the product of the total mass M times two of the distance ratios (a, b, or c). The two distance ratios used are the same two defining the products of inertia. If we are describing the product of inertia with respect to the x and y axes, then we add the factor Mab; for the product of inertia with respect to the y and z axes we add Mbc; *etc.* This redefining of the inertia matrix into new coordinate systems is often necessary since local transformations occur frequently in human body modeling.

The *Rinertia* procedure determines the inertia matrix of each rod and then adds the two matrices together. We are allowed to employ the principle of superposition since the moment of inertia with respect to a given axis of a body made of several of the basic simple shapes may be obtained by computing the moments of inertia of its component parts about the desired axis and adding them together [6].

2.3. Center of Mass Routine

To define the center of mass of a system of n particles, let M equal the sum of all the different masses:

$$M = m1+m2+\cdots+mn.$$

Then the center of mass is:

$$\begin{aligned} Xcm &= [m1\cdot x1+m2\cdot x2+\cdots+mn\cdot xn]/M \\ Ycm &= [m1\cdot y1+m2\cdot y2+\cdots+mn\cdot yn]/M. \\ Zcm &= [m1\cdot z1+m2\cdot z2+\cdots+mn\cdot zn]/M. \end{aligned}$$

In order to determine the center of mass of a system of two rods, we must treat each rod

as a particle. If we assume the rod has an even mass distribution and that all its mass is concentrated at the center of mass (of each rod) then we are allowed to treat each rod as a particle. (We only do this in computing the center of mass, not anywhere else.) The coordinate values (*xn*, *yn*, and *zn*) represent the current center of mass of each rod in the current coordinate system.

The two rods are connected at one of their endpoints, but we want to interpret the two together as one object defined in a local coordinate system centered at the object's center of mass. This is necessary because when we apply a force, and hence a rotation, the 'object' or system of rods rotates about the system's center of mass. The center of mass's coordinates must be (0,0,0) in the local coordinate system for the applied rotation to make any sense. Therefore, once this procedure determines the center of mass of the system, it defines a new local coordinate system with this new center of mass as the origin. The calculated center of mass serves as the translation vector to update all the points (the endpoints and center of mass) of the object in the new coordinate system.

2.4. Angular and Linear Momentum Routine

In the particular case of a rigid body rotating in three-dimensional space about a fixed point O, the angular momentum of the body about the fixed point O is $H_O=\sum r_i \cdot v_i \Delta m_i$, where r_i and v_i denote, respectively, the position vector and the velocity of the particle P_i with respect to the fixed frame xyz centered at O. For the case of a continuous and evenly distributed body, the Δm factor is treated as the constant total mass M and can be moved in front of the summation sign [6]. This procedure also determines the linear momentum, but it is not currently used by any of the 'forces' routines. It may prove to be useful in the future and is therefore computed.

3. Path Planning Techniques, Collision Detection and Avoidance

Since an astronaut's ability to move about in a three-dimensional environment is an important consideration, it is vital that TEMPUS be able to effectively accomodate such activity. The automatic path planning mechanism planned for TEMPUS is meant to generate the actual path to be used by the inhabitants of the environment. Path planning methods which seem most promising for the needs of TEMPUS are highlighted. The methods surveyed include configuration space and subdivision algorithms. The appropriateness of these approaches vis-a-vis the needs of TEMPUS are discussed. The design and implementation of a clearance detection mechanism is also presented along

with a description of how such a facility may be used in an interactive or automatic path finding mechanism.

3.1. Path Finding

Path finding has been explored in the context of robotics to control the motion of manipulator arms or mobile robots in an environment with known obstacles. Simply stated, path finding deals with finding a continuous path that avoids obstacles, from an object's initial position to a goal position [11]. We will examine two general methods for automatic path planning: configuration space and subdivision algorithms.

3.1.1. The Configuration Space Approach

The *configuration space* approach attacks the problem of finding a path for a polyhedron by reducing it to one of finding a path for a point. Reducing the moving object to a point greatly decreases the complexity of calculating clearances and collisions, thus simplfying the whole path finding process. The simplified problem remains a valid version of the original if the enviromental obstacles are *enlarged* to compensate for the reduced moving object.

To transform the moving polyhedron and the obstacle to their respective forms the following algorithm is used [26]:

1. A point is chosen on the moving object to represent the object. This point is refered to as the reference point.
2. The moving object is translated such that the reference point is at the origin.
3. Each vertex of the translated object is subtracted from each vertex of the obstacle.
4. The transformed obstacle is the convex hull (the smallest convex polyhedron) that contains the points created in step 3.

The extension to the obstacle (Figure 3-1) represents areas in which placing the reference point would result in a collision between the moving object and the obstacle. The asymmetry of the extension is due to the fact that if the moving object's orientation is fixed, the reference point may not approach the obstacle for equal distances from all sides before a collision occurs.

If the transformation is carried out completely, an environment such as the one shown in Figure 3-2 is changed to one resembling Figure 3-3. The transformed environment allows the find-path problem for polygon 'A' to be reduced to finding a

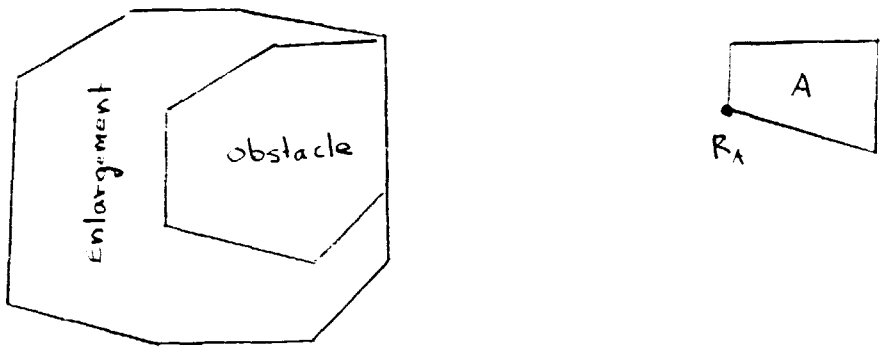


Figure 3-1: Enlarged Obstacle Due To Fixed Orientation of A [26]

path for point R_A around the enlarged obstacles. The shortest collision free path for the reference point consists of piecewise linear paths connecting the initial position and the goal position via the vertices of the extended object [25].

Using the obstacle vertices and the start and goal position as nodes, a graph of the environment can be formed. Two nodes are adjacent if they can be connected by a straight line that does not intersect any other obstacle. This approach allows the find-path problem to be solved via a graph search [25].

The configuration space approach does well in locating paths for objects whose motion is strictly translational (fixed orientation). The paths it finds are, however, very sensitive to inaccuracies in the object model. For an exact model, the calculated path would result in the moving object just touching the obstacles. The slightest inaccuracy in the modeling makes collisions almost certain [25].

In most practical applications of path finding the moving object is not restricted to translational motion. Rotational freedom permits the moving object to use paths that are not suitable for the strictly translational case. Figure 3-4 illustrates an example in which the goal is unattainable using translational motion alone; the object can reach its destination only when allowed to change its orientation.

The benefits of rotational motion are not without their cost. The environment's

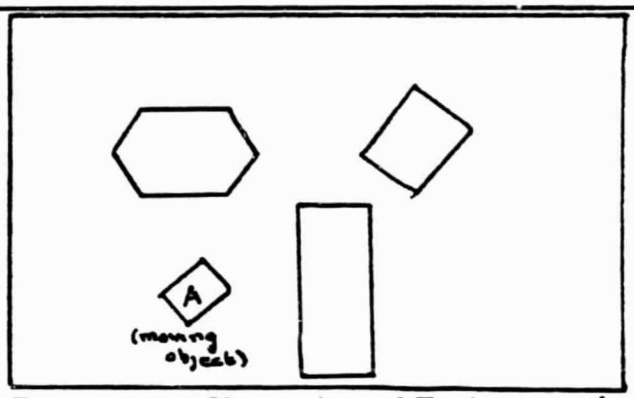


Figure 3-2: Untransformed Environment [25]

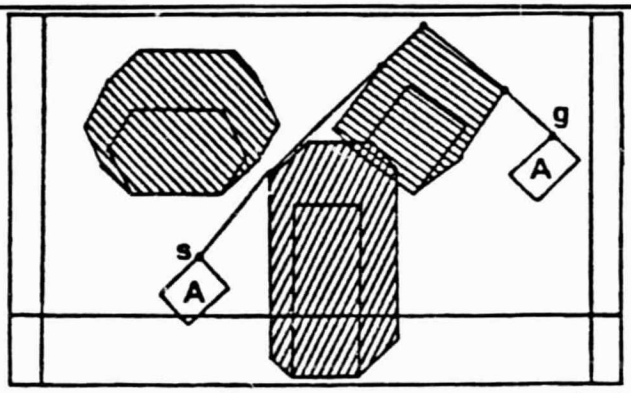


Figure 3-3: Transformed Environment [25]

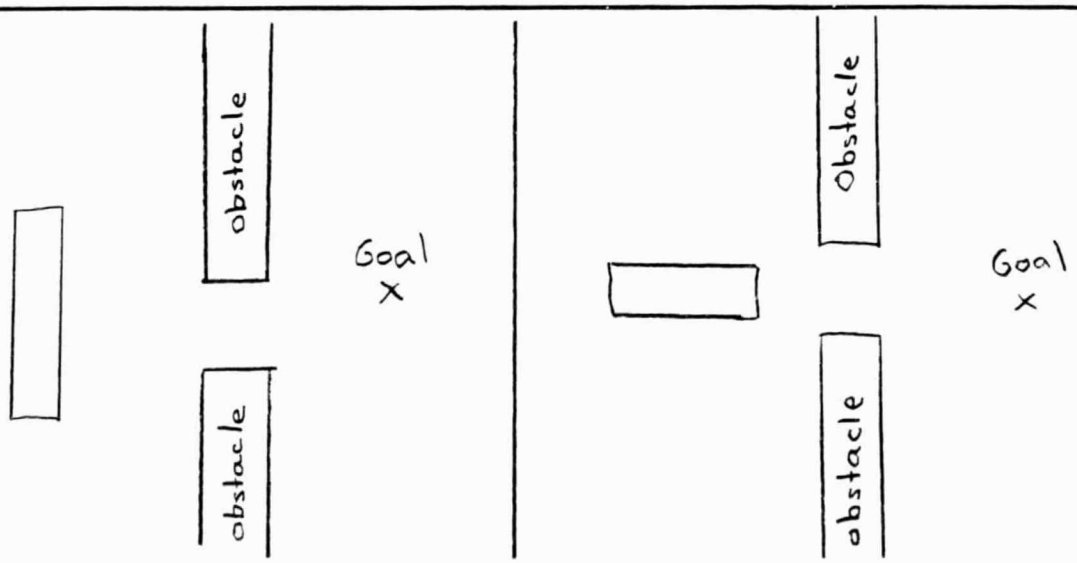


Figure 3-4: Advantage of Rotations

ability to accomodate the actual rotation and the new orientation must now be accessed

by the path planning mechanism. The actual rotation of the object refers to the area swept out by the object as it rotates to its new orientation [24]. This is not a trivial problem and it adds very significantly to the cost of path finding. Only very recently have sophisticated mathematical tools been applied to the complete six degree-of-freedom configuration space (three translations and three rotations) to determine paths for non-convex, three-dimensional objects [15, 17]. So far the results have only been extended to three degree-of-freedom, fixed base, robot manipulators in fixed (static) environments [27].

The configuration space approach handles rotations by creating new environments or "slices" for the rotational sweep and the object in its new orientation. Figure 3-5 shows how an obstacle will differ for each case.

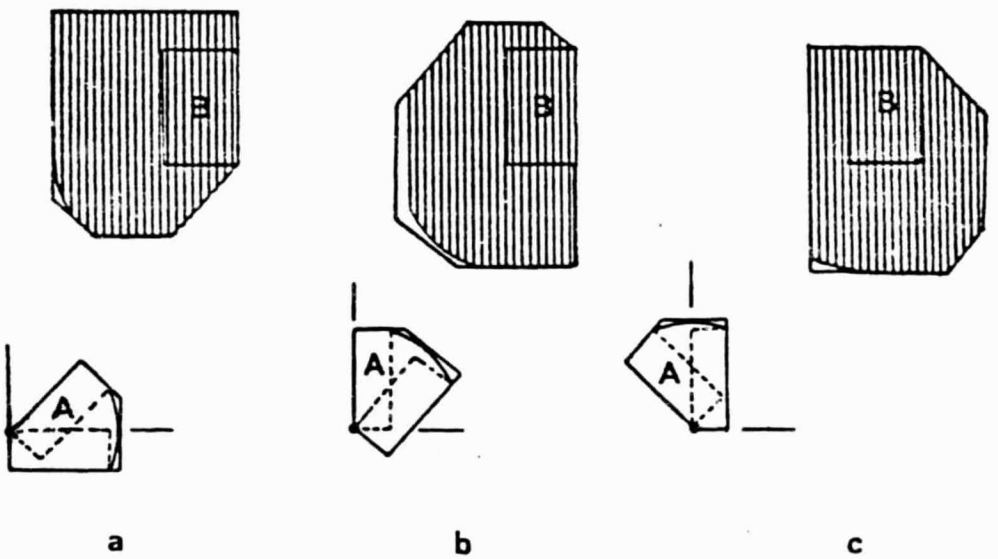


Figure 3-5: Obstacles Changing Due To Rotation [25]

When rotational motion is allowed, the environment is modeled by a series of slices. In reaching its destination the object may travel within a slice or between slices. The intra-slice motion represents translational motion while inter-slice movement signifies changes in orientation. An example of the slice approach is illustrated in Figure 3-6. Notice that inter-slice movement can only be done at points safe in both slices [26]. The intermediate slice corresponds to the object actually rotating to its new orientation.

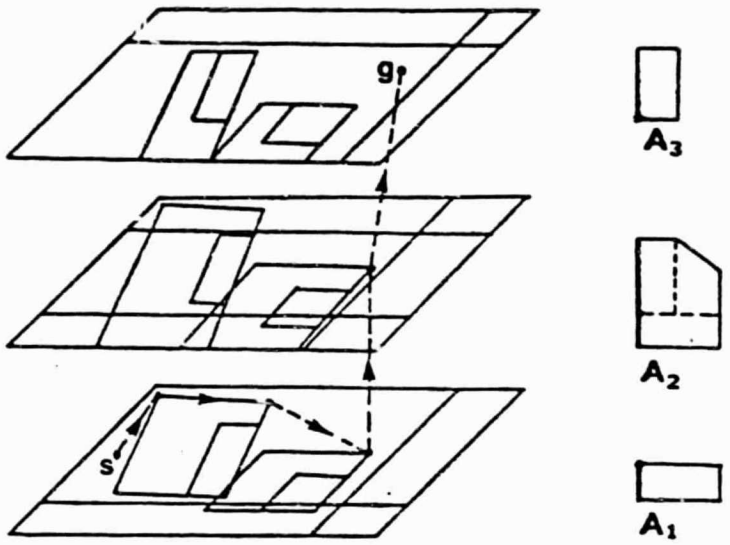


Figure 3-6: Path Finding Using Slices [25]

Although the slice approach provides a solution to path finding that accomodates rotational motion, the requirement that a slice be created for each sweep and each new orientation renders this technique very expensive when dealing with complex objects and environments. As previously mentioned, each rotation requires the creation of two additional slices (one for the sweep, and one for the new orientation). If an object has a single degree-of-freedom, even if the freedom is subdivided into only eight discrete positions (45 degrees), it may be necessary to create 16 slices to model the environment. If an additional polygon, with the same freedom, is attached to this object it may become necessary to create additional slices for each slice of the original object. For any given position of the original polygon the linked polygon is capable of 8 different orientations, thereby requiring 16 slices, and thus such a linked object may require 16^2 slices. The sixteen represents the number of discrete samples of each degree-of-freedom, multiplied by two for the number of slices needed for each different rotation. The exponent signifies the number of rotational degrees-of-freedom (1 for each polygon).

The actual number of slices created may be less than indicated by the formula, because efficiency measures can be used to rule out certain orientations. The example shows how quickly the upper limit of slices needed grows with the number of degrees of

freedom, and that without efficiency measures path finding using the slice approach may be an expensive prospect. The objects described above are simple by TEMPUS standards. The TEMPUS human body model has 38 rotational degrees-of-freedom. The complete representation of the configuration space for this body (not counting the flexible spine) would require at least 16^{32}, or about 10^{36} slices! As of now there are no efficiency measures that can be be applied to these objects, and there is no reason to believe that any scheme that could bring the number of slices created to a manageable number would itself be cheap. So at the present, the cost of using even the most established path finding mechanism for TEMPUS is prohibitive.

3.1.2. A Subdivision Algorithm

The *subdivision algorithm* treats the environment in the same way as the configuration approach. The moving object is reduced to a point and the obstacles are enlarged. The subdivision algorithm differs, however, in the way it goes about finding a path. It divides the environment into "cells," where each cell is then classified as *full*, *mixed*, or *empty*. *Full* signifies that the cell has no free space and is completely filled by an obstacle. *Mixed* implies a cell has some free space. *Empty* is used to denote cells which consist entirely of free space [12].

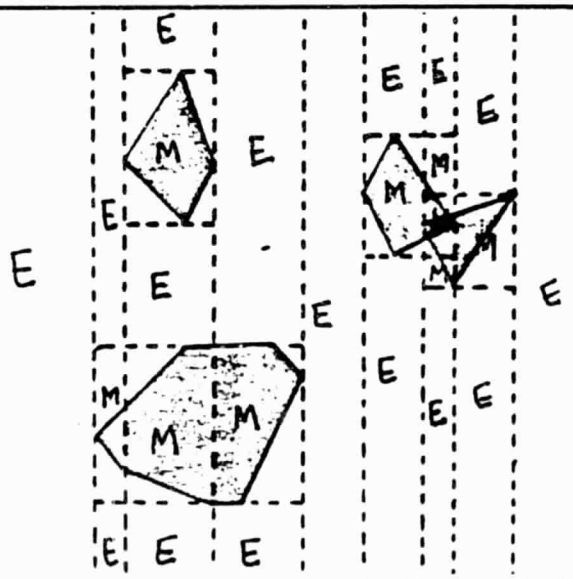


Figure 3-7: Enviroment Divided Into Cells [12]

After the environment has been divided the algorithm creates a connectivity graph (see Figure 3-8) where the nodes of the graph represent cells and are labeled E (empty), M (mixed) or F (full). There is an edge from each cell to its neighboring cell. The

algorithm attempts to find a path by traversing the connectivity graph via empty cells. If no such path is found the mixed cells are further subdivided and the graph traversal is attempted again. This process continues until either a path is found or the resolution (subdivision) reaches a limit set by the user [12].

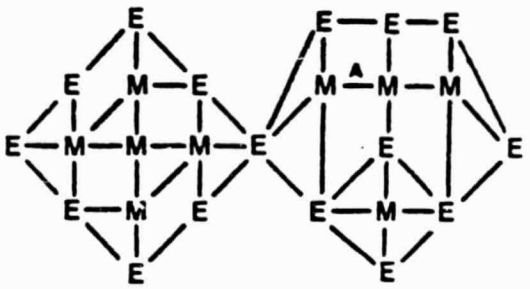


Figure 3-8: Connectivity Graph From Subdivision Algorithm [12]

The size of the cells represents a tradeoff. If the cells are made large, the resulting graph will be simpler but a greater number of iterations will be needed to find a path. For smaller cells the graph may be unnecessarily complicated but the algorithm will require fewer iterations. The size should be based on the environment. For cluttered environments small sizes are preferable; for open environments larger cells are adequate [12].

3.2. Path Finding in a TEMPUS Framework

The most interesting features of the methods discussed are the manners in which the environment is represented. The approach of converting the environment to a graph and then traversing the graph to find the path can be exploited in TEMPUS. The designers using TEMPUS know, for example, that certain areas represent corridors, hallways, hatches, *etc.* This knowledge can be used to facilitate path finding.

One possibility would be for the designer to create a master graph of the layout, consisting of areas connected by passageways or corridors. To get from a position to a

goal, the path planning mechanism would consult the master graph to see which passages connect the area in which the astronaut is currently situated to the area in which the goal is located. The mechanism would then find a path for the astronaut through his area to the corridor indicated by the master graph. The astronaut would then proceed via corridors to the goal area. Upon the astronaut's arrival at the goal area the path planner would find a path from the passageway through the goal area to the actual goal.

This scheme has the possibility of simplifying path finding in two ways. First, when the path planning mechanism is finding a way to a passage, or from a passage to a goal, it only needs to consider a subset of the layout. Therefore, the path finder has fewer obstacles to consider and the cost of enlarging, creating local graphs and traversing them is reduced. Second, since the corridors are designed for easy passage for the astronaut in a given position or positions, only specified slices need be considered. This is just an idea of how graph representation combined with knowledge of the enviroment may be used to attempt path finding. As of now no such mechanism exists.

The configuration space and subdivision methods represent the state of the art in path finding. They provide some direction in designing a path planning mechanism for TEMPUS, but in their given form they are inappropriate. These algorithms are inefficient even for simple non-jointed objects; for an application such as TEMPUS which deals with complex, multi-jointed objects with many degrees of rotational freedom their cost would be prohibitive.

3.3. Clearance Detection

A primitive path planning mechanism may be possible if it could ignore the complexity of TEMPUS objects and consider only approximations such as a bounding box or a convex hull of the object. This approach would greatly simplify path finding with the following consequences: any path generated would be for the approximated object, and the mechanism may not detect paths requiring TEMPUS objects to use complex motions such as joint rotation. Both these conditions can be managed if the paths produced are treated as recommendations and the user is given the tools necessary to evaluate the adequacy of the suggested paths.

Collision detection and clearance detection are tools that would be invaluable to a user trying to evaluate any path. These have been designed and implemented in TEMPUS. Clearance detection finds the closest components of two objects and the

distance between them. This information can help the user decide whether a figure can fit between two objects or if the fit is too tight. The user may also locate the narrowest spot in the path and test if the path is actually usable. Once a path is found the user may try people of different sizes in the path to get an idea of exactly how much room a path allows.

The applications of the clearance detection facility are not limited to primitive path finding mechanisms. It is extremely useful in other circumstances as well. In a more advanced path finding mechanism it may simplify the path finding problem because the information it provides can be used to limit the freedom of movement of TEMPUS objects. A great deal of freedom, as noted previously, complicates the task of path planning because of the substantial cost it imposes on the path finding computations.

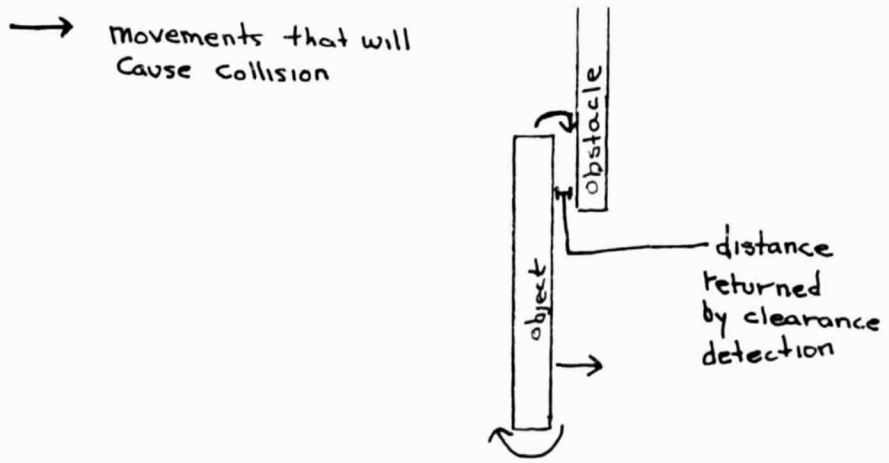


Figure 3-9: Using Clearance Detection To Limit Movement

If the clearance test indicates a component of an object to be very close to some obstacle, any rotational or translational motion that moves the component in question closer to the obstacle can be dismissed (see Figure 3-9). Relative to the path planning mechanism, the screening of certain motions is effectively the same as reducing the movement freedom of the object, thus simplifying the task of path finding.

The scope of clearance detection is not limited to path finding since it is very useful in other important TEMPUS functions such as reaching and evaluating object configurations. For a person in any given position TEMPUS allows the creation of a workspace: the polygon encompassing the set of points reachable by the person. If an

object is not within the workspace of a person the clearance test may be applied to calculate the distance between the workspace and the object (Figure 3-10). The result can be used to determine how the person should move next. For example if the astronaut is sitting in a chair and wishes to reach a switch, the clearance test can be used to decide whether the astronaut has to rise and walk or just lean forward.

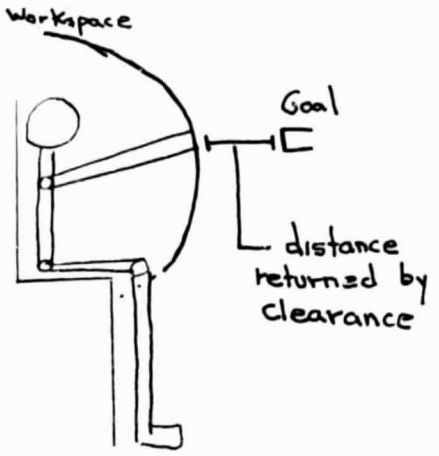


Figure 3-10: Using Clearance For Reaching

Further details on the TEMPUS clearance detection algorithm have been reported previously [2].

3.4. Interactive Techniques

While collision avoidance techniques can be used to determine initially plausible paths, and simple collision detection techniques can be used to ensure the validity of the exact derived paths, the overall computational complexity involved really forces us to adopt an interactive graphical approach to path planning and assessment.

Given the existence of adequate real-time graphical display capabilities we can substitute interaction for automatic planning. For example, three-dimensional view control can provide much (though not all) of a collision detection assessment: the operator can interactively and incrementally move the view to check for visual spacing

between objects. The only weakness is when one object has a concavity that is (partially) filled by another object. Then no global view will directly disclose the actual penetration; the only graphical recourse is to clip the objects by a plane passing through both, and visually check for intersection along a series of clipping planes throughout the concavity. This operation is made feasible by a feature on most real-time display systems to specify and manipulate a front clipping plane in real-time. In particular, the IMI 500 display purchased for NAS9-17239 has the required capability.

Although a bit more tedious, the clipping plane approach to collision detection may be extended to motion of one or more of the objects or figures along their respective paths of motion. Given a (pre-computed) path, for example, the object may be "flown" repeatedly along the path and visually inspected for interference by real-time manipulation of the viewpoint. Where a collision is suspected, the motion may be stopped and the configuration viewed in detail.

With the new generation of high-speed graphics workstations, it will probably become feasible to perform some partial collision detection "on-the-fly" during display update using a bounding box approach. The objects to be displayed have orthogonally-oriented rectangular prisms for bounding boxes which are not drawn on the screen. By incorporating a suitable spatial "hashing" scheme, neighboring boxes may be tested against one another to detect possible intersections. While the number of boxes used may be limited, it could be of significant help during interactive coarse motion planning.

The other principal advantage to a real-time interactive graphics system for motion planning is that the user would have full control over all the degrees of freedom of the figures involved. The substitution of human intelligence for (as yet unknown) zero-gravity motion heuristics could make the difference between a usable, successful system, and a slow, unresponsive planner. That is, the human operator would be able to manipulate the joints of the figures to achieve fairly close *positional* solutions in the extensive joint space of the human body. Using a real-time reach positioner based on six-dimensional inputs, the user could quickly experiment with potential body limb configurations while maintaining reseasonable real-time collision detection as noted above. The time needed to find a configuration by manual experimentation is apt to be significantly lower than that of any automatic method presently conceived. By computing distances between the user-controlled reach selection point in space and nearby surfaces, a semblance of "real" surface feel could even be achieved. In this

fashion it would not be possible to move the limbs into illegal intersections with the environment.

The system we would recommend for implementation would use interactive control and some planning heuristics. Since the automatic planning methods are expensive they would be suitable for coarse motion planning. Manual motion planning techniques, on the other hand, are computationally cheaper and more general, but are not be guaranteed to obtain a solution if one exists. Thus a balance between both is necessary, as the problem is practically intractable otherwise. In addition, the hybrid system may be easier to use and control by an experienced operator.

4. Data Collection Techniques

Many zero-gravity path heuristics used can be derived primarily through observation and analysis of films [4] of astronauts moving in zero-gravity (such as the Skylab films), scrutiny of their relevant comments, discussions with them about specific problems and techniques [29]. An important requirement is that the system should have a good subjective feel and its performance must degrade gracefully. The difficulty the system has in finding a path must reflect the difficulty an experienced astronaut in a known environment would have finding a path. For example, most astronauts appear to favor locomotion using the hands to maintain a grip and stability restraint, rather than using the legs or floating free.

In the next section will be look at an extensive proposal for an hierarchic simulation system which could take advantage of heuristics and knowledge of path planning as this information became available.

5. Proposal for an Hierarchic Simulation System

Complex processes such as zero-gravity human motion and task-directed activities pose certain problems for traditional simulation and reasoning techniques which emply a single level approach to the study of the nature of processes. In certain applications this single level view is entirely acceptable, especially when considering a microcosm of a large process or simply a limited process. In many other instances, however, processes are complex and naturally defined in terms of hierarchies. This hierarchical decomposition tends to organize the components of the simulation in a more effective manner. Unfortunately in traditional simulation, hierarchies are used strictly for organizational purposes. In this new line of research, we present more powerful abstraction hierarchies

which represent processes and objects using layers of nets each of which involves a valid simulation at a given level of detail.

The extensive proposal presented as Appendix I presents a definition of *hierarchical reasoning* about processes using different abstraction levels. Through hierarchical reasoning, the analyst is given much freedom in controlling the flow of actions for a given set of processes over an arbitrary number of levels.

A potential implementation of the hierarchical reasoning theory called *HIRES* is also presented. HIRES allows the user to reason in a hierarchical fashion by relating certain facets of the simulation to levels of abstraction specified in terms of actions, objects, reports, and time. High and low level knowledge about interacting, complex processes is integrated into a unified methodology.

This research is prompted by the need to adequately study human motion in a spacecraft environment. One of the major components of such a study is a simulation system that permits an analyst to model the environment, set up initial conditions, and then simulate some aspect of the environment. The study of articulated motion of human figures is sufficiently complex to warrant the use of the proposed reasoning system. Hierarchical reasoning methods may be employed to both reduce computational complexity during simulation and allow the analyst to better comprehend the processes being simulated.

6. Conclusions

The conclusion of this report is that zero-gravity motion studies should be undertaken in the context of a hybrid system combining interactive human positioning and movement with some automatic path finding techniques. The interactive part should be implemented on a real-time display system permitting the simultaneous control of at least six degrees-of-freedom of the human model. The resulting positions and motions should be visually inspected for collisions and interference. Path planning will initially be an interactive activity. As the interactive code is used, real-time geometric tests for limb reach and object collisions should be developed. If possible, these tests should be extended to handle object to object intersections at the graphics workstation code level.

While the interactive system matures, an hierarchic simulator should be

implemented both to enable more effective use of computational resources and to remove as much of the detailed manual positioning task as possible from the operator. This simulation system also provides a representational framework upon which known (or potential) movement heuristics may be implemented and tested.

Finally, the dynamics of articulated human motion must be completely defined for the zero-gravity environment and routines to simulate and compute the body's movement response to external or internal forces must be designed. This stage will necessarily include integration of human strength models in order to properly handle internally-generated forces. Suitable information on the allowable levels of reactive forces on environmental objects will also be required. The principal challenge here is to properly design the integrated system to permit effective operator interaction and control between kinematics and dynamics.

7. Schedule and Resources

The tasks outlined in the Conclusion could be realized over a four year period if suitable personnel were directed to its implementation. The schedule would, of course, differ if other directions were taken. In particular, a completely interactive system may take only two years, while a more elaborate path planning facility would take longer. It is assumed that appropriate real-time graphics display equipment already exists; such is the case on the current NASA contract. The approximate timetable for a zero-gravity motion studies system is given in Table 7-1.

The *time milestone* is the length of time from project inception (not a duration) to the completion of the indicated *tasks*. The tasks are a summary of the work needed to fulfill the system requirements discussed in the Conclusion. Each task refers to *one* graduate research assistant. This is a half time load (20 hours/week). Thus multiple tasks for one time milestone are assumed to proceed in parallel, and a total of two individuals for three years are required.

The resources required are summarized in Table 7-2. The monetary estimates are based on solely on 1985 University of Pennsylvania rates including employee benefits, tuition, and overhead as applicable. There is no provision for inflation; that may be projected by NASA as necessary.

Table 7-1: Zero-Gravity Motion Studies Schedule

Time Milestone	Task (per staff member)
year 0.5	Real-time motion playback. Representations for hierarchic simulator.
year 1	Real-time positioning. Elaboration of several levels of the hierarchic simulator.
year 2	Integration of playback/positioner with simulator. Simple force input for articulated body dynamics.
year 2.5	Refine simulator; determine and encode 0-g path heuristics. Determine forces from strength model and environment.
year 3	Experiment with real-time collision detection. Integrate dynamics model into TEMPUS.
year 4	Build motion planning system for coarse motion strategies. Integrate dynamics model into simulation and planning system.

Table 7-2: Zero-Gravity Motion Studies Resources

2 Graduate Research Assistants for duration of project.....$50K/year
Faculty supervision time (10% of academic year).....10K/year
Equipment:

Travel, current expense, duplicating, etc.....34K/year

Totals:
- Year 1: $94K
- Year 2: $94K
- Year 3: $94K
- Year 4: $94K

8. Bibliography

1. Howard Anton. *Elementary Linear Algebra*. John Wiley and Sons, New York, NY, 1977.

2. Norman I. Badler, Jon Korein, Paul Fishwick, Jeff Gangel, and Jane Rovins. TEMPUS: Simulating personnel and tasks in a 3-D environment. Progress Report #16, NAS9-16634, Dept. of Computer and Information Science, University of Pennsylvania, October, 1984.

3. Norman I. Badler, Philip Lee, and Sui Wong. Strength Modeling Report. Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1985. (For NAS9-16634).

4. Norman I. Badler. Motion Analysis Report. Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1985. (For NAS9-16634).

5. P. Bapu, S. Evans, P. Kitka, M. Korna, and J. McDaniel. User's guide for COMBIMAN programs. Univ. of Dayton Research Institute, January, 1981. U.S.A.F. Report No. AFAMRL-TR-80-91.

6. Ferdinand P. Beer and E. Russell Johnston Jr.. *Vector Mechanics for Engineers: Statics and Dynamics*. McGraw-Hill, New York, 1977.

7. F. M. Blakeley. "CYBERMAN". Chrysler Corp., Detroit, MI, June, 1980.

8. Mary L. Boas. *Mathematical Method in The Physical Sciences*. John Wiley and Sons, New York, NY, 1966.

9. John P. Boysen, Peter R. Francis, and Rex A. Thomas. "Interactive computer graphics in the study of human body planar motion under free fall conditions". *Journal of Biomechanics 10* (1977), 783-787.

10. Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez, Matthew T. Mason (Ed.). *Robot Motion: Planning and Control*. MIT Press, Cambridge, MA, 1982.

11. Rodney A. Brooks. Solving the Find-Path Problem by Representing Free Space as Generalized Cones. 674, MIT Artificial Intelligence Laboratory, May, 1982.

12. Rodney A. Brooks and Tomas Lozano-Perez. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. 684, MIT Artificial Intelligence Laboratory, December, 1982.

13. Rodney Brooks. Solving the find-path problem by good representation of free space. Proc. AAAI National Conf. on Artificial Intelligence, Pittsburgh, PA, 1982, pp. 381-386.

14. T. Calvert, J. Chapman, and A. Patla. "Aspects of the kinematic simulation of human movement". *IEEE Computer Graphics and Applications 2*, 9 (November 1982), 41-50.

15. John Canny. Collision detection for moving polyLedra. 806, MIT Artificial Intelligence Laboratory, October, 1984.

16. Jesus Depena. "Simulation of modified human airborne movements". *Journal of Biomechanics 14* (1981), 81-89.

17. Bruce R. Donald. Motion planning with six degrees of freedom. 791, MIT Artificial Intelligence Laboratory, May, 1984.

18. M. Dooley. "Anthropometric modeling programs – A survey". *IEEE Computer Graphics and Applications 2*, 9 (November 1982), 17-25.

19. J. T. Fleck, F. E. Butler, and S. L. Volgel. An improved three dimensional computer simulation of crash victims. DOT-HS-801, 507-510, Dept. of Transportation, NHTSA, April, 1975.

20. Cliff Frohlich. "The physics of somersaulting and twisting". *Scientific American* (September 1981), 155-164. (The Amateur Scientist).

21. Michael Girard and A. A. Maciejewski. "Computational modeling for the computer animation of legged figures". *Computer Graphics 19* (1985). to appear.

22. R. Harris, J. Bennet, and L. Dow. CAR-II - A revised model for crew assessment of reach. 1400.06B, Analytics, Willow Grove, PA, June, 1980.

23. E. Kingsley, N. Schofield, and K. Case. "SAMMIE-a computer aid for man-machine modeling". *Computer Graphics 15*, 3 (August 1981), 163-169.

24. James U. Korein. *A geometric investigation of reach*. MIT Press, Cambridge, MA, 1985.

25. Tomas Lozano-Perez. Spatial Planning: A Configuration Space Approach. 605, MIT Artificial Intelligence Laboratory, December, 1980.

26. Tomas Lozano-Perez. "Automatic Planning of Manipulator Transfer Movements". *IEEE Transactions on Systems, Man and Cybernetics 11*, 10 (October 1981).

27. Tomas Lozano-Perez. Colloquium. (at University of Pennsylvania, 1985).

28. Tomas Lozano-Perez and Michael Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles". *Comm. of the ACM 22*, 10 (October 1979), 560-570.

29. J. R. Morrow Jr., J. Boelter, W. W. Hosler, and J. Jackson. Man-machine analysis of translation and work tasks of Skylab films. Final Report, NAS9-15521, NASA JSC.

30. M. R. Ramey and A. T. Yang. "A simulation procedure for human motion studies". *Journal of Biomechanics 14*, 4 (1981), 203-213.

31. D. H. Robbins, R. O. Bennett, Jr., and B. Bowman. User-oriented mathematical crash victim simulator. Proc. 16th Stapp Car Crash Conf., 1972, pp. 128-148. SAE Paper No. 720962.

32. Micha Sharir and Elka Ariel-Sheffi. On the Piano Movers' Problem: Various Decomposable Motion Planning Problems. 58, Dept. of Mathematical Sciences, Univ. of Tel Aviv, and Computer Science Dept., Courant Institute Of Mathematical Sciences, New York Univ., Feburary, 1983.

33. Paul C. Shields. *Elementary Linear Algebra*. Worth Publishers, New York, 1980.

34. Jearl Walker. "The essence of ballet maneuvers is physics". *Scientific American* (June 1982), 146-153. (The Amateur Scientist).

35. Jane Wilhelms. The simulation of animals, robots, and other articulated mechanisms using dynamics. Dept. of Computer Science, University of California, Berkeley, CA, 1984.

I. Hierarchical Reasoning

This Appendix consists of Paul Fishwick's PhD Dissertation proposal. The pages are separately sectioned and paginated from the body of the Zero-Gravity Movement report. A Table of Contents for this Appendix is therefore included.

Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction

Paul A. Fishwick

Dissertation Proposal

May 1985

Table of Contents

List of Figures

Abstract

This thesis presents a definition of hierarchical reasoning about processes using different abstraction levels. Most simulation (or process reasoning) systems permit the monitoring of a single level of abstraction. Through hierarchical reasoning, the analyst is given much freedom in controlling the flow of actions for a given set of processes over an arbitrary number of levels.

An implementation of the hierarchical reasoning theory called HIRES is also presented. HIRES allows the user to reason in a hierarchical fashion by relating certain facets of the simulation to levels of abstraction specified in terms of actions, objects, reports, and time. High and low level knowledge about interacting, complex processes is integrated into a unified methodology.

Acknowledgments

– to be completed –

Chapter 1
Introduction

1.1. Overview of this thesis

This thesis presents a new theory for hierarchically reasoning about processes involving complex actions and a variety of objects. In reasoning hierarchically, the analyst is able to view the simulation model as a hierarchy of abstractions in terms of processes, objects, time and reports. A sample implementation of the theory, called HIRES (HIerarchical REasoning System), is also presented. Interactive sessions and example simulations are given for a simple alarm clock model and a more detailed spacecraft environment including moving human operators.

1.2. Problem Statement

Traditional simulation techniques and reasoning methods have employed a single level approach to the study of the nature of processes. In certain instances this single level view is entirely acceptable, especially when considering a microcosm of a large process or simply a limited process. In many other instances, however, processes are complex and naturally defined in terms of hierarchies. This hierarchical decomposition tends to organize the components of the simulation in a more effective manner. Unfortunately in traditional simulation, hierarchies are used strictly for organizational purposes. In this new line of research, we present more powerful abstraction hierarchies which represent processes and objects using layers of nets each of which involves a valid simulation at a given level of detail.

This research is prompted by the need to adequately study human motion in a spacecraft environment. One of the major components of such a study is a simulation system that permits an analyst to model the environment, set up initial conditions, and then simulate some aspect of the environment. The study of

articulated motion of human figures is sufficiently complex to warrant the use of the proposed reasoning system. Hierarchical reasoning methods may be employed to both reduce computational complexity during simulation and allow the analyst to better comprehend the processes being simulated.

1.3. Recent Related Work

The theory and implementation of the hierarchical reasoning paradigm discussed in this thesis owes much to the past research in the areas of general simulation, knowledge representation, and abstract reasoning (such as qualitative reasoning). Many of these important efforts are discussed in chapter 2. To put the new concept of hierarchical reasoning into proper perspective, a subset consisting of several recent key research areas is briefly discussed. Then in the next section the specific unique contributions of this thesis are outlined and contrasted against the previous work.

1.3.1. Qualitative Reasoning Systems

Qualitative Reasoning involves a high-level of reasoning about a given process. Expert knowledge and heuristic information are used to form the semantics of a system based on qualitative reasoning. This type of reasoning is often used to understand the nature of processes and is to be differentiated from the type of reasoning utilized in the development of semantics for lower level classical simulation systems.

There are several qualitative reasoning systems such as [de Kleer 79, Forbus 81a] to cite specific examples. DeKleer's system allows the analyst to reason about electrical circuits. DeKleer's other work includes *envisioning* which represents predictions on the sequence of events that can occur at a given time, and a qualitative theory of differential equations in [QR 84]. Forbus' system (called FROB) allows the user to reason about a "bouncing-ball" environment using a set of high order, abstract semantics. Forbus terms this new set of semantics "Qualitative Process Theory" which is discussed completely in [QR 84].

It is important to note that qualitative reasoning represents a methodology for reasoning about processes at a single level of abstraction. Although, this new, higher

level of abstraction is critical in the comprehension of processes in general, the theories involved with strict qualitative reasoning do not allow for multiple levels of abstraction. Nor do they allow for the ability to choose various levels of abstraction during the course of a given process simulation. Hierarchical reasoning, on the other hand, permits multiple levels of abstraction during the simulation and gives the user methods for focusing on different aspects of the simulation at different abstraction levels.

1.3.2. STEAMER

STEAMER [Hollan 84] represents a substantial development in terms of allowing an analyst to reason about a steam propulsion system in terms of graphical icons and displays. In the example simulation of the steam plant, there are roughly 100 pictures that represent a hierachical description of the system. The pictures are accessed in a hierarchical fashion, since the user may wish to view the system as a whole (i.e. the basic steam cycle) or in parts.

The hierarchy present in the objects within STEAMER represents an object hierarchy and not a process hierarchy. More specifically, there is a single level of abstraction in terms of the actual process. The user is allowed to view various levels, but these different levels rely on the underlying mathematical level which controls the simulation. The simulation of the steam propulsion system does not rely on the type of qualitative semantics commonly found in qualitative reasoning systems. The results viewed by the user can be qualitative, but the underlying, computational model is not.

1.4. Contributions of this Thesis

1.4.1. Identifying types of abstraction

This thesis contributes the notion of hierarchical reasoning in terms of a basic theory, an implementation, and representative examples. The theory of hierarchical reasoning is based upon three primary types of abstractions:

1. *Object Abstraction* - The view of an object can take many different forms depending on the level of abstraction associated with the object. If the

object is large (relative to an agent's size) and can be explored from within, then the object can be internally described as being composed of various partitions. If the object is small, then the object can be described using either coarse (i.e. bounding volume) descriptions or more specific descriptions (i.e. surface description or solid-geometry model). The object hierarchy within STEAMER provides a good example of allowing the user to view object abstraction from a graphical perspective.

2. *Process Abstraction* - The view of a process can be seen at a fairly low level, using mathematical terms, or at a high level (using qualitative reasoning).

3. *Report Abstraction* - The view of the simulation can be filtered by having only certain levels of process abstraction output to the user.

1.4.2. Goals

The primary goals of this thesis are:

- Create a taxonomy for abstraction. The general notion of abstraction as it relates to process simulation and reasoning is described in specific terms. Types of abstraction are presented. The relationships among each type are explored.

- Create formal definitions for process and object abstraction. These definitions will include the method of hierarchically reasoning about processes in that specialized hierarchies of processes and objects are described.

- Create an implementation to allow a user to exercise the various abstraction concepts defined.

1.5. Terminology

The term "hierarchical reasoning" is used since one is able to reason about processes by taking advantage of the inherent abstractions of processes, actions, and objects. Hierarchical reasoning is a different type of reasoning than qualitative reasoning in the following way: qualitative reasoning involves the description of a set of semantics to be used in high-order reasoning about processes, whereas hierarchical reasoning involves a set of capabilities that allow the user to control the flow and structure of processes. Therefore qualitative reasoning deals with defining semantics for a single abstraction level. Hierarchical reasoning deals with defining methods for monitoring and controlling the flow among many abstraction levels. Some of the

high-order process levels used when reasoning hierarchically may in fact be defined using the semantics commonly found in the qualitative reasoning literature.

1.6. Overview of Chapters

Chapter 2 presents the general areas of study that impinge upon the proposed area of research. The areas of human motion, computational geometry, robotics, simulation, and planning are briefly covered.

Chapter 3 describes the primary application to be used to describe the abstraction concepts: EVA (Extra Vehicular Activity).

Chapter 4 defines the concept of hierarchical reasoning. The notions of abstraction and a taxonomy for abstraction are outlined. The relationships among the abstraction types are enumerated in the form of a triad.

Chapter 5 defines a formal model for process abstraction.

Chapter 6 defines a formal model for object abstraction.

Chapter 7 presents an initial implementation of some of the abstraction concepts in the form of a message-passing system. The function of an alarm clock is used as an example.

Chapter 8 discusses some preliminary conclusions and contributions of this research. Problems discovered during the work and suggested courses for further research are outlined.

1.7. Background

A few years ago, an interactive software system called *TEMPUS* [Badler 83a]was initiated. The major goals of TEMPUS were to create a set of advanced graphics capabilities, anthropometric databases, and body modelling functions to aid in the study of human motion. Recently there has been work done in the areas of user interface [Weinstein 83, Bloom 83] and reach kinematics [Korein 84]. Currently, an

animation system is in the planning stage and should help in creating film and video animation of human models.

Much of the underlying framework necessary for the study of human motion has been developed during the course of the TEMPUS project. The project has a strong 3-D CORE [CORE 79] implementation, several graphics utilities [Shapiro 84], and significant low-level geometric facilities [Korein 84]. A new research direction concerning the simulation and planning of multiple agent activities within a space (zero-gravity) environment is underway. This new work will rely on the foundation created through prior years on the TEMPUS project. The new research, however, is concentrating mainly on issues whose bases lie within the artificial intelligence domain. Specifically, the areas of parallel and hierarchic planning, collision avoidance planning and simulation based on a complex 3-D world model are being pursued. The world model will include a set of objects and actions whose effects will be constrained by the zero-gravity environment within the workstation.

1.8. Applications

This research is prompted by NASA Johnson Space Center Contract NAS9-17239 and Army Research Office Grant DAAG29-84-K-0061. The primary goals of the work includes the study of human motion within a space environment. Such an environment might include a shuttle or orbiting space station where several complex on-board and extra-vehicular tasks are performed during the mission.

Chapter 2
Background Areas of Interest

This chapter covers several areas that are relevant to the proposed theory of hierarchical reasoning. These areas are not a direct part of the proposed research, however, they serve as background areas so that a foundation may be laid for the theory and implementation of hierarchical reasoning in future chapters. All areas are active research areas that have been studied for the past fifteen years or so. The areas that we will review will be the following:

1. **Human motion** - This is clearly the focus of the project's overall research efforts and therefore one of the primary areas of interest. Much of the work done within this area, however, has not addressed itself to the simulation and planning aspects of a true human motion system.

2. **Computational Geometry** - At the bottom of all of the high-level functions that we wish to possess, we find the geometric objects and human models themselves. The quantitative attributes of a model, such as a space station and its contents, must not be sacrificed when the motion modeling is underway. Our motion system, for instance, should be able to tell us when something is **physically** impossible or impractical.

3. **Robotics** - Humans move in a way not unlike the way that robots move (at least theoretically). The robotics domain provides equations of motion that can be applied to human joints and appendages so that we may better understand human limb movement. Human movement is much more complicated due to the articulation and spherical joints of the human body, however, many of the basic motions of both robots and humans may be compared if we do not analyze the joint movements to a great level of detail.

4. **Classical Simulation** - Classical simulation techniques are explored (i.e. both discrete and continuous approaches). These systems represent the first study of simulating and reasoning about processes.

5. **AI Simulation** - Simulation systems constructed within the AI domain are presented. These systems are more recent than the classical simulation systems, and they are more concerned with representing and manipulating symbolic knowledge in addition to quantitative knowledge.

6. **Planning** - What is the most efficient method of achieving task A, given a number of initial conditions and a goal? Planning can help us to determine an efficient method of accomplishing a task.

2.1. Human Motion

There has been a fair amount of work done in the study of human motion. Much of this work, however, has not dealt with simulation or planning to any great level of detail. Some of the earlier work was based on Labanotation [Hutchinson 70]. Labanotation is a notation for human movement which uses symbols to describe the movement and orientations of the joints over time. Labanotation *scores* are much like musical scores in their time representation attributes. Early human movement work often entailed the construction of an interpreter for Labanotation [Smoliar 77]. Other work such as [Calvert 80] discussed a mixture of labanotation techniques with analog interpretation of body movements (using an electrogoniometer).

Badler described a taxonomy for conceptual movement descriptions [Badler 75] (including human motion) and also co-authored a survey article dealing with human movement representations [Badler 79]. A recent publication discusses the incoporation of dynamics into human movement [Badler 83b]. Dynamic effects yield a more accurate simulation of human motion than has been previously explored. Zeltzer describes a method for describing the *motor control* of various skeletal parts [Zeltzer 82] using finite state automata. He uses the broad jump and a short walking sequence as examples of skeletal motion. Zeltzer also mentions possible directions in human motion simulation using a production system model [Zeltzer 83]. Weber [Weber 78] defines a set of types that might be used in a human movement simulation system. Tsotsos [Tsotsos 80] has designed a system called *ALVEN* which includes frame-based knowledge of ventricular heart motion.

There has been little work in applying AI techniques to the problems found in the human motion domain. The work of Badler [Badler 75] includes descriptions of various motion representations using frame knowledge, however, there is much more work to be done. The research under the NASA project at the University of Pennsylvania is currently concentrating in this area.

2.2. Computational Geometry

If one is to have a reliable simulation or planning system, it is essential that the geometry of the environmental components be considered. Many of the examples given in AI literature tend to ignore geometrical considerations when describing such things as *moving blocks* and *picking objects*. To some extent, this is quite understandable, given the difficulty and computational overhead in precisely defining an accurate geometric simulation. Nevertheless, we consider the geometrical aspects of the simulation to be important if one is to maintain a reliable system.

Collision detection has long been a topic of research with its primary applications in the robotics domain. See section 2.3 for a discussion on the relationships between the study of robot movement and human movement. Intersection methods using planar projections and a three-dimensional octree object representation are discussed in [Ahuja 80]. The work of Lozano-Perez [Lozano 79] and Brooks [Brooks 83a, Brooks 83b] is important for its contribution to collision avoidance and hence collision detection. We note that collision detection is closely related to collision avoidance, however, this latter topic is explored in the section on planning.

Korein [Korein 84] has recently written about the computational aspects of *reach* and its implications in a workstation environment. The swept volumes created through various reach movements are also outlined. These volumes can be used as a basis for studying collision detection between human limbs in motion.

The concerns of computational geometry involve low-level, quantitative aspects of processes. Should a simulation of human movement include such quantitative details? It depends on the type of *process abstraction* necessary to adequately study a particular sequence of movements. A complete discussion of *process abstraction* can be found in Chapter 5. If we are investigating the reach spaces associated with a tightly confined environment such as a cockpit, then it becomes important to simulate these computational aspects. On the other hand, it may be entirely inappropriate to simulate low-level sweeping motions or collision detection situations if we are modeling astronauts conducting an EVA (extra-vehicular activity) in a wide-open space. It might turn out that a simple set of heuristics will detect collisions, etc.

The notion of *abstraction* in terms of processes, objects, and reports represents the primary avenue of research that is defined in this thesis.

2.3. Robotics

When researching human motion, we may take advantage of the enormous amount of work that is being done in the robotics field. Many of the "production" robotics languages are procedurally oriented. Any interactions between manipulators or devices is strictly encoded using concurrency measures such as semaphores and monitors. This makes sense, considering that the manipulators controlled through the language have to operate in real-time. The robotics languages are not so much oriented toward simulation as they are toward accomplishing real-time tasks. Examples of these robotics languages include AL [Mujtaba 82] and AML [Taylor 82]. A fairly high-level language called AUTOPASS [Lieberman 77] was designed which contained task descriptions as language statements. A survey of 14 robot languages may be found in [Bonner 82].

Many of the concepts, such as **grasping**, **ungrasping**, and **reaching** are similar in both human movement and robotics: our simulation system can reap the benefits of many of the low-level computational aspects often found within robotics literature. Despite some of these similarities between the robotics and human motion worlds, there are some striking differences. The primary difference lies with the fact that robots are currently pre-planned to perform certain tasks: we give a robot a plan or set of instructions and fully expect the robot to operate accordingly. Humans, when doing formal tasks, may also be considered to operate in a "pre-planned" mode although humans may spontaneously react to various external conditions that arise during the execution of the task/plan. The point is that we program a robot in a certain way because we already know what we want it to do, whereas, we need to **study** human movement because we are trying to understand it. Another difference is in the degree of locality associated with the robot's actions. Many robots are composed of one manipulator which works in a localized region (possibly performing a pick and place or assembly operation). Humans have a tendency to move about, so modeling human motion has some inherent complexities which are not found when modeling the movements of robot manipulators.

Due to the strict procedural nature of the production robotics languages and the lack of a "simulation" flavor, the production robotics languages are not seen as being appropriate when studying human motion. The *AI Simulation* approach is based on a more non-procedural foundation. In the *production system* approach, for instance, there is **one** monitor which controls the flow of information between the nodes in the system. Each node might have procedures which may be activated, but they do not pass control to other procedures directly. Using an object-based approach, one is capable of invoking a large class of objects at one time through message passing primitives. In either approach, one is not forced into a strict procedural framework.

2.4. Classical Simulation

2.4.1. Simulation as an Analysis Tool

In simulating a particular world, such as the world within an operating manned spacecraft, we learn more about the internal processes within that world and how they interact. Simulation is a subject which has been studied longer than most others mentioned so far in this report.

2.4.2. Two Categories of Simulation

Classical simulation is currently thought of as being in one of two categories:

- **Discrete** - Discrete simulation languages allow studying a model with discrete time events. Events occur at a specific time, and when the world model time has reached the event time, the event "fires" and a world state change is effected. Many of the business-oriented simulation languages support discrete event simulations since time can be effectively modeled with discrete state changes.

- **Continuous** - The scientific and engineering environments often must simulate world that involve gradual state changes. For instance, if one is simulating the trajectory of a moving projectile, then it becomes necessary to break time into very small intervals until some tolerance is satisfied. When the interval size is too large it is shortened for more precision.

2.4.3. Discrete Simulation Languages

The early simulation languages, such as SIMSCRIPT [SIMSCRIPT 72] and GPSS [GPSS 67] are essentially discrete event simulations systems: time is suddenly incremented according to event arrival times. It also should be noted that the knowledge represented about the world in such systems does not have a symbolic treatment. The manipulation of knowledge is somewhat limited by the expressive power of the language implementation: most of the major simulation languages, such as GPSS, are FORTRAN lookalikes. SIMSCRIPT supports a more English oriented development system and seems to be geared toward being able to create a simulation quickly. Because the older simulation languages have evolved over a period of many years, there are some quirks. For instance, SIMSCRIPT supports many synonyms for accomplishing the same simulation sub-task (such as queuing an event). This attribute of SIMSCRIPT can be annoying - in a sense, it is trying to be a pseudo natural language interface, but it fails since the user is often forced to remember which are valid synonyms and which are not. A cleaner, more precise syntax would have been more appropriate.

2.4.4. Continuous Simulation Languages

Continuous simulation languages allow a variable break-up of time over small intervals. The interval chosen is based usually on a tolerance that is required for an accurate simulation. CSMP [CSMP 76] is a good example of a pure continuous simulation system. The simulation user constructs a *model* of some physical structure using a network of interconnected blocks. Each block represents a function with corresponding inputs and outputs. Then a set of equations describing the physical system is constructed and translated into the block network. The block network acts as an analog representation of the physical system. Blocks are of certain types, such as ***integrators***, ***summers***, ***weighted summers*** and ***relays***. Throughout the simulation, the user may optionally plot certain variables over time, and collect other relevant statistics.

2.4.5. A Combination of Continuous and Discrete Methods

There are some simulation languages, such as GASP [Pritsker 74] which support both discrete and continuous simulation techniques. Payne [Payne 82] gives a good overview of classical simulation systems and includes a section on combined techniques. This combination is important to the NASA work. Discrete simulation affords us the capabilities of stepping through time in chunks. Many of the NASA tasks may be thought of as discretely defined tasks:

1. Open Inner Hatch
2. Turn Rotary Switch-14 45 degrees
3. Extend to Stow
4. Grasp tool-2

A question arises: why could not some of these tasks be considered continuous events? Monitoring time *continuously* generally means that we start by dividing time into very small intervals of the same size. If the intervals do not give us the required tolerance necessary for accurate results, we can adjust the time increment. The answer to the above question is that any event may be considered from either a *continuous* or *discrete* vantage point. The method that we should apply depends on where we want to concentrate our simulation results. For instance, suppose we are modeling an astronaut that is performing a complicated manuever to pass through an airlock. Let us further suppose that the motion is "complicated" since there is an obstruction which must first be removed before gaining entry through the airlock. When the astronaut gets to the other side of the airlock, he will perform Task-A (whatever that may be). If we are interested in modeling Task-A, then we may very well treat the complicated motions of going though the airlock as a few simple discrete events. On the other hand, if we are more concerned with the method used to navigate through the airlock, we may choose a more "continuous" treatment of time around the airlock entry and a more "discrete" treatment of time during the execution of Task-A. We see that this discussion is related to the idea of *process abstraction* mentioned in the introduction. Specifically, the user should be able to

"tune-in" on certain parts of the workspace, and "tune-out" other parts which still need to be simulated, but at a coarser level. The proposed research design includes the capability with which one may fully specify the *process abstraction*, *object abstraction*, and *report abstraction* inherent within a system (see chapter 5).

2.4.6. Problems with Classical Systems

The world model that is supported by both continuous and discrete simulation languages is often described using data structures that are indigenous to languages such as FORTRAN. Many of these simulation languages do not support a symbolic structure of the world model, therefore reasoning about the model can be difficult. The next section describes simulation techniques that are amenable to working with knowledge at many different levels.

2.4.7. AI Simulation

2.4.7.1. Quantitative and Qualitative Reasoning

Let us now consider some of the more recent developments with respect to simulation. Much of this recent work is to be found within the context of AI literature. One may divide the study of current simulation systems into two areas: *qualitative* and *quantitative* simulation.

In reasoning *qualitatively* about a system, we utilize certain heuristics about the world model. This type of reasoning involves an abstract, intuitive sense about the world. Let us take an example using the everyday world. If we drop a ball on the ground then it will probably bounce. We can therefore set up a causal relationship between **drop ball -> bounce ball**. The ball environment may be found in [Badler 75, Forbus 81a]. We see that this knowledge is very important since it represents the way we think about the world in an everyday-sense. When I drop a ball on the ground, I do not calculate parabolic trajectories to determine that it will bounce - I know that it will bounce from past experience. Also, there is a fairly good possibility that I will know the general path that the ball will follow. It should be noted that this example presumes a world model with a gravitational force. The space model will also contain similar heuristics about both the motion of objects and human agents.

Most of the early pioneering work in qualitative reasoning may be found in [de Kleer 75, de Kleer 77, Rieger 77] who demonstrate the use of qualitative reasoning in solving various physics problems. Forbus continues to explore this arena [Forbus 81a, Forbus 81b, Forbus 83]. The STEAMER project [Hollan 84] is also an excellent example of representing and manipulating qualitative knowledge. The interested reader is directed towards [Gentner 83] and more recently to a special issue in qualitative reasoning [QR 84] which contains a wealth of literature dealing with items such as the semantics of qualitative process theory and causal reasoning.

In reasoning *quantitatively*, we view the world through a mathematical perspective. We are concerned with seeing the world according to the equations that have been derived to explain certain phenomena such as differential equations of dynamic motion, and trajectories for path finding. Regardless of whether or not the equations give us *intuitively* correct results, we tend to trust their accuracy since they are based on many years of research and empirical knowledge.

Both types of reasoning are important to a simulation system. Quantitative reasoning is imperative in certain instances, such as in the simulation of an autonomous device such as a robot arm (or a human arm). For instance, the dynamics associated with a robot arm performing an assembly task become critical due to the real time nature of the manipulator task. Therefore, if we are modeling an assembly robot's arm movements, we might well choose a quantitative modeling method. Qualitative reasoning will permit us to include *rule of thumb* production rules in our space-environment simulation. Expert rules (the experienced astronaut being the expert) will be embedded into our simulation to aid in qualitatively reasoning about the space workstation. Much of this reasoning cannot be defined in a quantitative sense (or when it is defined as such, it becomes overly complicated and meaningless). For instance, by **grasping** a given tool, I can reason that this tool will stay held within my hand as I move around. To define this using a **strict** quantitative approach, we lose the very idea of "grasping". Grasping can be considered to be a qualitative aspect of a simulation - it is a symbolic, abstract piece of knowledge.

2.4.7.2. STRIPS

The STRIPS system [Fikes 71] was one of the first AI simulation systems and serves as a general model for many of the currently known systems. STRIPS is a robot problem-solving system. The three basic concepts (state of the world model (SWM), production rules, and control monitor) are an integral part of STRIPS. The SWM consists of a list of *assertions*. The inference rules are in the form of *operators* which manipulate the world model. A simple example of a robot operator is **stack** which may be defined as follows (using an example from [Nilsson 80]):

```
stack(obj1 obj2)

  Preconditions:  HOLDING(obj1),CLEAR(obj2)
  Delete:         HOLDING(obj1),CLEAR(obj2)
  Add:            HANDEMPTY,ON(obj1,obj2),CLEAR(obj1)
```

This operator may be defined in English: "Stacking of one object (**obj1**) onto any other object (**obj2**) may begin only when the robot is holding **obj1** and **obj2** has nothing on top of it (i.e. it is clear)." The "preconditions" are assertions that happen to be in the state of the world model at a given instant. That is, when HOLDING(obj1) and CLEAR(obj2) are within the SWM, then the **stack** operator is free to "fire." The "delete" items are deleted from the world model after the operator fires, and the "add" items are added. In this particular case, the delete and precondition rules are identical, however, that is not always the case.

2.4.7.3. Recent Simulation Research

The STRIPS model was an excellent start, and there have been several major simulation efforts since. One of the major problems with STRIPS is that it only treated time in a discrete way. In fact, time as a continuous phenomenon does not exist within a STRIPS model. The most notable effort around the same time as STRIPS is a system developed by Hendrix for modeling continuous concurrent processes [Hendrix 73]. His *Scenario* concept is directly related to the production rules in STRIPS although he goes much further. His most important contributions

are the capabilities for modeling gradual changes in processes (such as the flowing of water) and for allowing multiple processes to interact, sometimes in un-scheduled ways. Among several examples that Hendrix discusses, "filling the bucket" is a classic example of these concepts. Even though the general flavor of the simulation methodology is oriented towards *discrete* simulation, the gradual filling of the bucket with water may be *continuously* modeled through the calculation of a set of time intervals for gradually updating the water level inside the bucket. The simplest method of finding the small time intervals necessary for a continuous simulation is done by including not only the original time-dependent motion equation, but also an equivalent inverse equation dependent on the motion variable (i.e. displacement, for instance). This *filling of the water bucket* is a scenario which operates independently of the rest of the world. However, if an outside agent (such as a robot) turns the valve on the water spigot, the scenario is able to detect such an "interrupt" and adjust itself.

An implementation of the Hendrix Model was achieved by [Lowrance 77]. Lowrance and Friedman include several different examples including an electric switch world and a Turing machine world.

Salter [Salter 80, Salter 84] and colleagues have recently developed a simulation system which is strongly based on the Hendrix simulation model. The system, called CONCUR, permits the user to create a world model composed of *scenarios* and then simulate the world. CONCUR contains an equation solver which solves a set of equations in order to arrive at the *interesting times*. The *interesting times* for a particular process are those time values which may be determined by solving for the inverse solution to an equation dependent on a time variable (such as an equation of motion). CONCUR also includes a powerful pattern matching capability which aids in the instantiating of process scenarios.

Vere [Vere 83] built a simulation based on the application of planning the motions of an autonomous spacecraft (such as Voyager). Vere's work is more important for its contribution to parallel planning in time, so it will be discussed in section 2.5.

While on the subject of simulation, it is worth discussing *animation*. Animation

might be used as an important feedback within any simulation system. It allows the user to visually interact with the simulation - changing the environment and seeing the results. In the NASA project, we are interested in maintaining a considerable amount of visual feedback so that the user of the simulation can feel more in tune with the simulated world. It is important to note that the system that we are in the midst of designing should not be thought of as only an *animation* system. Any animation that is desired can easily be achieved by including a scheduled event that fires every fraction of a second. Our primary goal within our research is to understand the interrelationships between human agents and the rest of the world model. Animation techniques serve a useful purpose in allowing us to carefully monitor such interrelationships in terms of changes to the state of the world model. Other feedback mechanisms include numeric output such as statistics, variable values, and possibly question-answering in a limited form. In the chapter on hierarchical reasoning, the reader will note that animation of a process can often be associated with a fairly low level of *object abstraction*.

The STEAMER Project [Hollan 84] is a good example of a system that was designed to allow the user to reason with it. The user is also capable of creating his own simulations by way of a graphical editor. Thus, STEAMER has placed an emphasis on user-interaction and consequently a strong animation capability.

Fantasy-Model simulations such as ZORK [Lebling 79] and STARCROSS [STARCROSS 82] (a science-fiction spacecraft simulation) are also interesting since they allow the user to communicate to the simulation system in natural language. The user becomes part of the simulated environment, learning about the world by trying out various actions and receiving responses to those actions. The "fantasy" component may easily be translated to any other domain, including the spacecraft domain where the user is trying to learn about a spacecraft by exploring it. Fantasy simulation games may be categorized as a very high-level discrete simulation where the user is able to invoke functions at each clock tick.

2.5. Planning

2.5.1. An Overview of Planning Methodologies

Planning and simulation are the two most important aspects of our current research. By finding a plan for an agent, we are trying to determine the sequence of events that will allow him to achieve a given goal. So according to this definition of planning, we recognize the notion of "optimality" within planning. That is, planning is something that we do so that we can determine the most efficient path (or action sequences) before we actually perform the task. Here we see a classic relationship between planning and simulation. We may be inside a simulation system and decide that we need to develop a "plan" before continuing. We then execute some sub-module of the simulator called the "planner". Once we receive a plan, we simulate it. Planning may therefore be seen as a sort of "mental" simulation within the actual simulation. Within the planning system, we must practically simulate a given action just to determine the plan. For instance, if we review the production system simulation method, we find that we perform the same basic computations for both simulation and planning. The major difference is that with planning we work in reverse, starting from the goal(s) and working backward towards the initial conditions. Planning, in this respect, is analogous to "reverse" simulation.

Let us define two new terms: *automatic planning* and *manual planning*. They will be defined as follows:

1. Automatic Planning - A procedure to determine a plan given a goal and a set of initial conditions. The goal is often expanded using a backward-chaining procedure until the set of initial conditions are satisfied. In many instances, branches of the state space may be pruned when it is determined that the branch will never (or probably never) reach the initial state. Automatic planning will often be referred to simply as "planning" in this report since planning, as it is usually described, is defined as an automatic procedure.

2. Manual Planning - A plan that is "manually" derived by the user of the simulation system, or by someone who has statically encoded some kind of plan knowledge within the simulation. This has also been termed "deterministic" simulation. In this thesis, manual planning will take the form of a *task function*. The task function will simply be a deterministic function containing control structures and hooks into the knowledge database. This is the type of planning that Hendrix [Hendrix 73]

mentioned in his simulator. One simply arrives at a representative function and runs it through the simulation system.

The type of planning capability that one should use depends on the situation. As an example of these two types of planning, consider the following two situations:

1. **Opening the hatch door** - Opening the hatch door to the module, for instance, enables the agent to move from one area to another. However, the act of opening the door does not generally require a pre-conceived plan. Opening the door might well be decomposed into sub-tasks and control structures. If our knowledge of opening hatch doors is good, then this procedure can be straightforwardly programmed as a *task*.

2. **Moving from the Module to the Orbiter** - Moving from one place to another is often somewhat complicated by obstacles which might be in the paths of the agents. To find efficient paths, it is useful to generate *automatic* plans. The necessity for the automatic plan is derived directly from the complexity inherent with a given action. The act of moving can involve complex conditions.

If we are modeling the actions of robots in the environment, we may very well choose to incorporate a strictly automatic planning approach. On the other hand, humans will not always move in accordance with a fixed plan. It may be worthwhile for the simulation system user to develop a plan then try it out to see what happens. It is clear that both *manual* and *automatic* planning are important to a simulation system.

2.5.2. Creating Plans with Time Constraints

The efforts of STRIPS led to linear methods for determining plans, but Sacerdoti [Sacerdoti 77] created the first parallel planner (NOAH) which was able to create a set of parallel plans that were *partially* ordered in time. A conflict resolver was added to correct any discrepencies in planned orderings of events. Even though Sacerdoti's work included partial ordering, there were no specific time intervals or durations associated with the events. The treatment of **time** using more specific time constraints is important, but only recently researched. Vere [Vere 83] has created a system called DEVISER that creates parallel plans for controlling the actions of an autonomous spacecraft. DEVISER is important for its emphasis on *planning in time*. Goals and various actions/events may have *window* or *duration* attributes. A window in the system is characterized as follows:

(WINDOW EARLIEST IDEAL LATEST)

The last three variables represent, respectively, the earliest time an action/event can occur, the ideal time for it to occur, and the latest possible time that it can occur.

Salter [Salter 83] has also written about planning with time constraints. His system deals with time in terms of *time trajectories* which can be seen to be somewhat isomorphic to Vere's *window*. The time trajectories are reminiscent of timing diagrams for integrated circuits: State values vs. time.

Allen [Allen 81, Allen 83] has written about representing time intervals and general temporal reasoning. Temporal reasoning is related to temporal planning since one is concerned with representing networks of time durations and intervals.

2.5.3. Collision Avoidance Planning

Within the robotics domain, collision avoidance planning [Lozano 79, Brooks 83a, Brooks 83b] enables a mobile robot to avoid obstacles in its path. Fixed robots may also have to worry about obstacles in the way of the arm. For the purpose of simulating human motion, these collision detection methods will become useful. However, it is hoped that an adequate simulation can be accomplished by applying certain movement heuristics to a given path problem in the spacecraft. It remains to be seen what the problems in this area will be. In any case, collision detection may need only be implemented at a fairly high level, such as modeling continuous motions in a packed environment. The notion of *process abstraction* arises again.

Chapter 3
An Application: EVA

The research for this thesis will use a specific application within the spacecraft domain so that the concepts to ensue will be better comprehended. The chosen application is the process of an EVA (Extra Vehicular Activity) as performed on a spacecraft such as the Space Shuttle. An EVA, as its definition suggests, involves operator movement outside of the spacecraft. Early EVA's were performed with the use of a tether (life line): the astronaut would simply exit the vehicle and perform observations with camera equipment. More recent EVA's are used not only for observation but also for the diagnosis and repair of the external part of the spacecraft or orbiting satellites. The modern EVA is accomplished with the aid of an MMU (Manned Maneuvering Unit) which permits the operator to travel with ease.

A general overview of an EVA may be found in [Joels 82], but the most critical and comprehensive information is to be found in the actual checklists used by operators within the spacecraft [Armstrong 83]. A "checklist" is the set of procedures which is used by human operators to insure the correct and safe operation of tasks delineated within the checklist. Each checklist will contain several activities which may contain sub-activities some of which might be contingent upon certain variables. Thus, a checklist is very much a program whose statements are English-like.

The use of a NASA checklist to describe the characteristics of abstraction is quite appropriate when we consider the complexities associated with the lists: there are literally hundreds or perhaps thousands of steps within a checklist. The EVA checklist, in particular, is used for the following reasons:

1. Spatial Aspects - The agents involved within an EVA must move around the craft to perform various preparations prior to exiting the vehicle. During the EVA, there are many spatial considerations since the agents must work both inside and outside the craft.

2. Abstraction - Every type of abstraction is needed due to the complexity of the EVA. Movement must occur via the airlock and each type of movement may be simulated at several levels. An interesting abstraction feature (which will be discussed later at some length) relates to the view of the spacecraft by the agents. From within, the spacecraft is a space defined by specially identified locations (i.e. ***in the commander's seat***, ***in the airlock***, ***in front of hand controller***). From the outside, the spacecraft is viewed externally as a solid object. The "spacecraft" object is still the same object in both instances, but the methods of reference and levels of *object* abstraction differ.

The EVA procedure [Armstrong 83] may be abstractly represented as shown in figure 3-1. It can be seen that this procedure involves a definite hierarchy. Also certain tasks may be performed in parallel with other tasks so that the overall procedure is efficiently defined. As an example, we note that the MIDDECK PREP and AIRLOCK PREP procedures may be performed in parallel.

EQUIP PREP
  MIDDECK PREP
  AIRLOCK PREP

EMU CHECKOUT
  EMU 3 CHECKOUT CONFIG
  SOP CHECK
  PRIMARY REGULATOR/FAN/PUMP CHECK
  EMU 3 TEMPORARY STOWAGE
  COMM CHECK
  BATTERY CHARGE CHECK

EVA PREP
  PREP FOR DONNING
  EMU DONNING
       CHECK
       PURGE
  COMM CHECK
  EVA COMM CONFIG
  UHF ONLY SITE COMM CONFIG

DEPRESS/REPRESS
  DEPRESS
  REPRESS
  FAILED LEAK CHECK (5 PSI)

POST EVA

POST EVA ENTRY PREP

EMU MAINT/RECHARGE
  WATER RECHARGE
  OXYGEN RECHARGE VERIFICATION
  SUIT DRYING/SEAL LUBRICATION
  WATER RECHARGE VERIFICATION
  EMU LiOH CARTRIDGE/BATTERY REPLACEMENT
  BATTERY REPLACEMENT (Crewman in Suit)
  LiOH REPLACEMENT (Crewman in Suit)

EVA CUFF CHECKLIST

EMERGENCY AIRLOCK REPRESS

Figure 3-1: EVA Checklist

Chapter 4
Hierarchical Reasoning

In this chapter, we describe the central ideas of the thesis and define exactly what is meant by the phrase "hierarchical reasoning." We shall see that hierarchical reasoning is based on hierarchies of different abstraction types. These types will be explored in detail in the remaining chapters. Before describing the various aspects of abstraction, we discuss the need to perform research in this area.

4.1. The Need to Explore Hierarchical Reasoning

So far, in our background discussions, we have outlined much of the research connected with simulation and planning. It is clear that our NASA work can benefit greatly from this past experience. The NASA work, however, poses some new challenges which are not addressed by prior systems. The overall NASA project must address the following items (during the course of this new line of study):

1. **Multiple Agents** - Even though some work has been done in dealing with multiple agents [Konolige 80], this is essentially an untapped area. Simulating the cooperation of agents in performing a single task will become important. We can rely on the wealth of knowledge in operating systems theory [Dijkstra 68, Hoare 78] to help with the study of the cooperation and synchronization of processes (i.e. agents acting on objects).

2. **A Complex World Model** - A true simulation of a space environment will undoubtedly involve several agents and many objects and actions (perhaps hundreds or thousands). The simulator should be able to handle complex models **efficiently**.

3. **Human Agents** - There has never been a realistic simulation of human agents in a working environment. There have been some definitions of simple movement functions, but this does not qualify as a true simulation (this previous work is seen more in the light of a *functional decomposition* of certain high level tasks).

Hierarchical Reasoning

This thesis will not attempt to address the many issues previously defined as background material. Instead, a specific topic is proposed: the design of a new simulation method which is more appropriate to complex simulations. The primary feature of this method lies within the notion of hierarchical reasoning which is now defined.

The previously discussed simulation and reasoning systems are useful for certain applications, but they fail to provide the analyst with a capability of integrating abstract levels of knowledge about the simulation environment. If we take the example of a bouncing ball [Badler 75, Forbus 81a], we see that we can describe the motion of the ball qualitatively: that is, we can break the general space of the environment into regions and define causal relationships such as DROP BALL -> BOUNCE BALL. On the other hand, using a quantitative approach, we could use a set of equations to specifically point out the precise position of the ball over time. It is clear that both approaches, including the pieces of knowledge associated with each approach, need to be integrated in a single knowledge base if we are to correctly represent the ball object and all its facets of motion (at all levels of abstraction). In this manner, the analyst at simulation time could choose the level of abstraction as a function of variables (and ranges of those variables) such as spatial location, time, and object class (assuming a hierarchical object representation).

Abstraction in simulation was discussed in [Goldin 81], however their approach was oriented towards the automatic, bottom-up creation of higher-level "chunks." A formal methodology for abstraction was not presented. STEAMER, which was discussed in the introduction, does include the capability for the user to reason in terms of different levels of object abstraction. STEAMER, however, does not focus on process abstraction or the topic of abstraction in general: it is more concerned with providing the user with a detailed graphical interface to the model.

Hierarchical reasoning centers around the ability to monitor and change abstractions associated with the model. Three major types of abstraction are discussed: process, object, and report. The focus of the hierarchical reasoning methodology is not to advocate a particular set of semantics for a given abstraction level (such as qualitative process semantics or graphics). In fact, different levels of

abstraction will utilize some of these semantics. Rather, the focus is to give the simulation user the flexibility in both manipulating and monitoring the simulation by working with the inherent abstractions associated with the model. Hierarchical reasoning, therefore, is concerned more with the flow and control of the simulation and less with specific operational semantics associated with different levels.

4.2. Definitions

Hierarchical Reasoning is defined as reasoning about processes over levels of abstraction. The term "abstraction" is the key concept when reasoning about processes hierarchically, so we will spend some time delineating the aspects of abstraction. *Abstraction* is an often-used word and can be found strewn throughout general literature. It is a word that we use to define our representations of things such as objects and actions. That is, objects and actions may be described "abstractly" or "non-abstractly." When one speaks of "abstraction" it is generally meant to convey a ***high level*** of abstraction, whereas more detailed accounts of things are not usually termed "abstract." In this thesis, we refer to abstraction as being an all-encompassing term which may have associated levels so that one may easily differentiate between various representations. Therefore, abstraction denotes the type of description and semantics connected with a virtual continuum that spans many levels.

4.3. Modeling Abstraction

Despite the fact that abstraction has been defined in many different instances within computer science literature, there has not been an investigation into a definition which encapsulates the essence of several different sorts of abstraction such as procedural, object, and report abstraction. In subsequent chapters, we form a clear view of abstraction in terms of definition and the modeling of systems at varying abstraction levels. It is not sufficient to simply talk of abstraction in a general sense: the various types of abstractions and their interesting relationships are explored.

4.4. Multi-Level Abstraction

Abstraction must be considered within the context of many levels. If one discusses abstract notions for processes and objects at only single level, then much relevant information will be missing. It is very natural to reason about processes at varying levels depending on the context in which the reasoning is being performed. In this thesis, we will be concerned with the definition of the many levels and the connections between the levels.

4.5. Computational Efficiency

We also concern ourselves with the efficiency of simulating certain processes at different levels of abstraction. Simulating a given process at a high level of abstraction will generally involve less computational complexity than if we were to simulate at lower levels. There is a trade-off between the level of abstraction and the information quality obtained at a given level. That is, we might be able to drastically reduce the complexity of a given task by simulating at a high level, however, we might lose valuable information about the process. If this information is not needed, then there will not be a problem. On the other hand, if the information is essential to the integrity of the simulation, then trade-offs will have to be made. These trade-offs and considerations are studied in this thesis.

4.6. Overview

We need to carefully define abstraction before any theory or implementation may be pursued. In this chapter, we discuss abstraction of the following types:

- Process
- Object
- Spatial
- Temporal
- Report

After defining these types of abstraction, we explore methods for modeling abstraction. Implementation considerations are left for a future chapter. We are now primarily concerned with developing a theory for abstraction.

4.7. Types of Abstraction

We now taxonomize abstraction by presenting several types along with accompanying definitions.

- *Process* - When dealing with actions, we may be interested in a fairly high level of process abstraction: If we are simulating the movement of an agent traveling though an airlock, we may simply want to specify the action as three simple discrete events: 1) Agent enters airlock, 2) Agent is inside airlock, and 3) Agent exits airlock. In a similar fashion, the EVA may be seen as 1) Preparation, 2) actual activity and 3) Post EVA considerations. On the other hand, if we are very interested in reasoning about a process in greater detail, then we will want to specify a fairly low abstraction level. Animation, for instance, will generally require lower-levels of process abstraction, unless we are willing to settle for widely separated key frames.

- *Object* - Objects can be defined as "things" at the highest level of object abstraction, or by such attributes as coordinate triples and orientation angles at lower object abstraction levels. Sometimes an object can be viewed as its bounding volume (a minimal volume that covers the object): for instance, during coarse motion planning of some simple geometric sort, the bounding-box object description might be all that is necessary. At other times, such as in an animation, the object must be treated at a low-level of abstraction. Note that space, in general, can be treated as a hierarchy of objects. Therefore, spatial locations may also be abstracted.

- *Spatial* - As was just mentioned, the abstraction of space is important so that we gain a locative understanding of an agent's movements. An agent might be simply "inside the crew compartment" or, at a lower spatial abstraction level, be "at middeck, next to the hatch." At still lower spatial abstraction levels, specific coordinate data may be necessary.

- *Temporal* - Time may be treated using varying-sized intervals. Large intervals can be used when a process is to be simulated at a high abstraction level. Time may also be treated using high-level phrases such as in [Allen 81] (X occurs before Y,etc.). For more accuracy and more detailed information, smaller time intervals will need to be specified.

- *Report* - Report abstraction relates to the procedural abstraction level at which the analyst wishes to "see" the results of the running simulation. The analyst may wish to run the entire simulation at a fairly low-level but view only higher level results. Using report abstraction, the analyst can filter out excess output.

4.8. Relationships among Abstraction Types

Now that types of abstraction have been identified, it is essential to look at the relationships which bind them. First, we create a pictoral relationship in the form of a triad (as seen in figure 4-1).

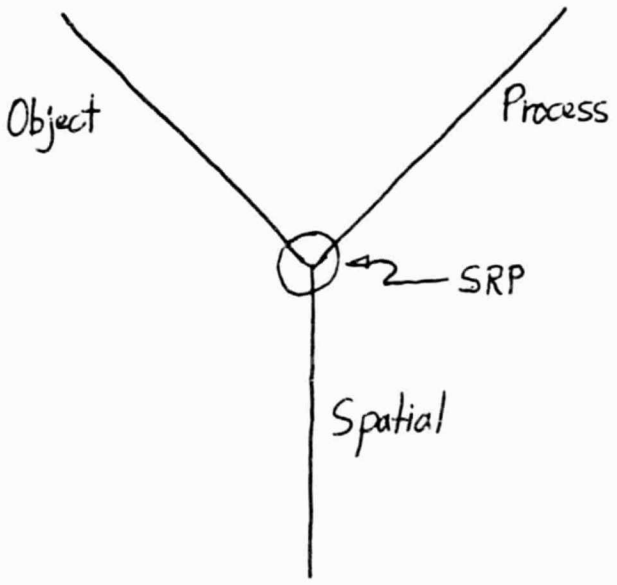


Figure 4-1: Basic Abstraction Triad

4.8.1. The Basic Triad

An explanation of figure 4-1 is in order. First, the acronym SRP denotes the Spatial Reference Point. The SRP can be seen as the vertex where process and object abstraction "merge" to become spatial abstraction. The SRP actually represents the point at which a given agent operating within a process starts to treat objects and processes from a spatial perspective. The triad is meant as a general paradigm for viewing the relative functions of each arm of the triad (object, process, and spatial abstraction).

The top of the triad represents fairly high levels of abstraction while the bottom represents lower levels. We note that the spatial abstraction segment is composed of both object and process abstraction. Let us examine each of the 3 segments starting at the upper left (object abstraction) and working in a clockwise direction.

4.8.2. Pure Object Abstraction

This relates to an agent's external view of an object. Objects such as a car or the space shuttle can be abstracted from an external point of reference by removing the degrees of detail associated with the object's exterior. A car from a distance may be abstracted as a rectangular box. As one approaches the car, the exterior begins to take on further amounts of detail. The space shuttle may equally be seen in a similar light: its abstraction might be seen as a cylinder. As one approaches the shuttle, one sees that it actually has four main components: a tail section, fuselage, wings, and nose.

4.8.3. Pure Process Abstraction

When one considers the nature of processes from a high abstraction level, processes can be seen as partially ordered sequences of events that have no direct spatial relationship. For example, we view the EVA as being composed of the three aforementioned events: 1) Preparation, 2) EVA, and 3) Post-EVA. These events are so high-level that when we reason about them, we tend to consider them as simply a series of events and not only as rough spatial abstractions. The spatial component of process abstraction is more emphasized at lower levels of process abstraction.

4.8.4. Spatial Abstraction

As we traverse throught the abstraction levels we come to the SRP which denotes an interesting transition. Specifically, objects are now treated from internal points of view. No longer do we see an object as a container of some sort, but rather the object becomes the space in which we operate. The kinds of abstractions that we can associate with a given object are greatly changed once we enter the object and move around inside. Every object has a SRP and it is clear that abstract information connected with the SRP depends on the agent. If the agent is a human and the object is a spacecraft, then it is quite logical to see that we can traverse the many abstraction levels associated with this object. On the other hand, if the object is a stowage-locker, then there is little chance that we will move down beyond the SRP of the locker. If we cannot get inside of the object, or have little interest in identifying it as a space, then we tend to traverse abstraction levels strictly above the SRP for that

object. The type of action used by the agent is also important when considering spatial abstraction for an object. If an action involves fine motion planning then we are more likely to identify a greater number of discrete spatial points of interest.

When we consider process abstraction, we see that all processes may eventually be reduced to simple movements at some abstraction level. The act of "donning the EMU" (Extravehicular Mobility Unit - a special "space suit") can be completely described by a partially ordered graph of translation and orientation events. Complicated movements of an agent through a space can be defined by a long sequence of connected vectors (This path could be calculated using a "findpath" [Lozano 79] algorithm or some other relevant technique).

4.8.5. Complementary Representations for Spatial Abstraction

As we have seen, both object and process abstraction merge to become spatial abstraction. The spatial abstraction segment may therefore be seen as being composed of two segments: the object side and the process side. We now consider these two sides to spatial abstraction.

On the object side, we can imagine a model describing the object's spatial aspect. The model (which will be discussed more completely in a subsequent section) can be defined as a connected graph of places or locations. The nodes of the graph represent locations and the arcs represent directional movements which allow one to traverse from one place to another. This technique has been widely used to define space qualitatively. Kuipers TOUR model [Kuipers 76] is an example of such a scheme. The TOUR model allows one to find and describe places, paths, and directions in a map domain (i.e. finding a geographical route).

On the process side, we can imagine a model describing the elements of fine motion details. We use Petri Nets in a future section to model these motions. These nets are very similar to the object nets with the exception that the process nets are naturally oriented toward defining processes rather than locations. The nets are composed of conditions (or places) and transitions.

When we examine these two sides to spatial abstraction, we see that both sides are

somewhat complementary aspects of the same notion. An object representation shown in great detail may be seen to be a "static" set of locations with connecting direction arcs. Spatial abstraction can be defined as the reprsentative data structure derived by taking a starting location and closing it under the operation of an action verb. Object abstraction which defines space is therefore a *state space* representation. The process representation contains similar information within the content of the model, except flow of control and constraints are emphasized rather than static locative data. We will discuss spatial abstraction more completely in a future section.

4.9. Defining Key Abstraction Types

We have identified three abstraction types and their relationships: 1) process, 2) object, and 3) spatial. From the above discussion, we see that we can limit the discussion to simply process and object abstraction since spatial abstraction can be seen to be a combination of these two types.

In consideration of temporal abstraction, we see that time is actually somewhat of an artifact and may be more adequately studied with respect to the partial ordering inherent within the events of a process. As more events are used to describe a certain process, the intervals of time used to capture the events become smaller in size. The actual time is important only in that it allows us to reference the current state of a process. Thus, when we pursue the essence of process abstraction, we cover temporal abstraction due to the proportionality between process and temporal abstractions.

The last type of abstraction (report) will be covered in a future chapter. There we will look at controlling the system of abstract levels that we are defining in this chapter. Report abstraction will allow us to filter information associated with processes.

Chapter 5
Process Abstraction

In defining processes at different levels of abstraction, it becomes necessary to first mention current methods for specifying process abstraction. It will be shown that these methods revolve around an incomplete theme which fails to adequately model process abstraction. Then we proceed to define a new model which is an improvement upon the older process model in that it covers abstraction more comprehensively.

5.1. What is a Process?

Processes can be complex, so we restrict ourselves to study simple models of processes, suspending discussions of the practically infinite variety of process description languages (such as general purpose programming languages). A process is construed to be a partially ordered graph of events. In short, a process is composed of actions. Some actions may occur in parallel with other actions (which introduces the partial ordering), and most actions will have certain constraints imposed upon them (the actual ordering or synchronicity).

5.2. The Traditional View of Process Abstraction

Processes may be viewed from qualitative or quantitative points of view (as described in the background chapter). The traditional view of a process is generally defined in figure 5-1.The view in figure 5-1 shows a single level of abstraction. This is the basic flaw of the traditional process view. In looking more closely at the figure, we see that even though abstraction is given treatment, its treatment is essentially denotational in character. PREP means that three preparations need to be performed (EQUIP PREP - Equipment Preparation, EMU CHECKOUT - Checking EMU, and EVA PREP - Preparations for actual EVA). The term PREP *denotes* these three

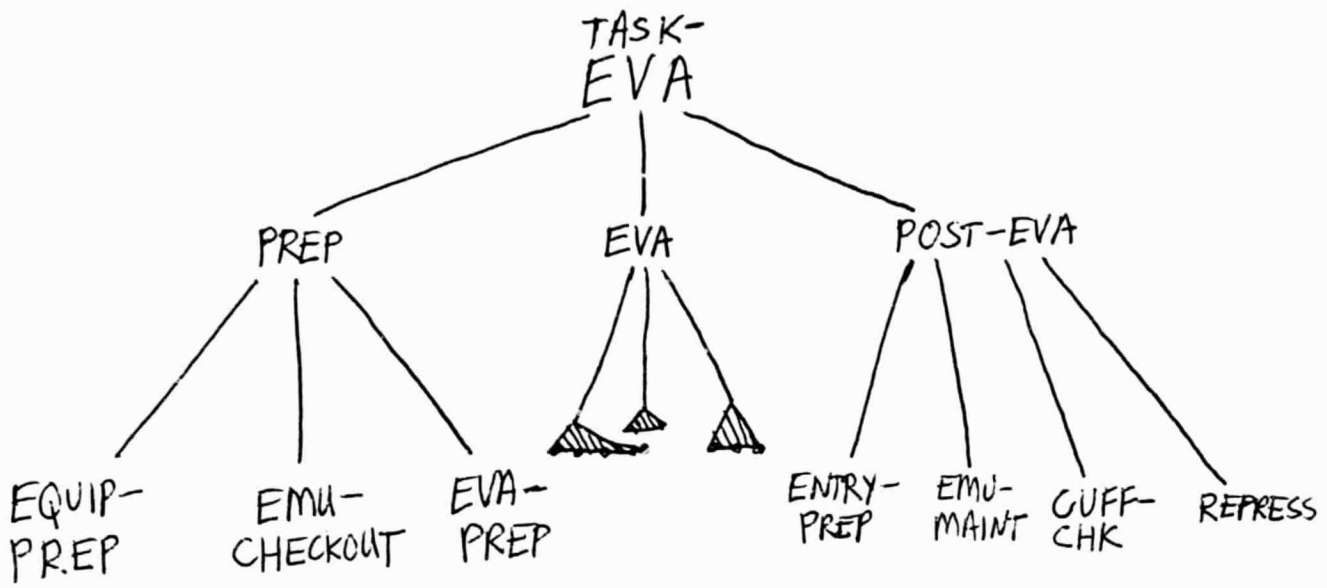


Figure 5-1: A Process Hierarchy

sub-processes. The key point to be made here is that PREP by itself has no intrinsic value other than the denotational one imposed by the tree. That is, PREP does not mean anything by itself: it simply serves as a kind of corridor though which we must pass to get to the "real" process. The real process is not actually uncovered until we execute the leaf nodes of the tree. While it is true that these internal tree nodes have provided with us with abstract process knowledge, it should be noted that this representation fails to give us any flexibility in reasoning about the overall process at different abstraction levels, since only the lowest levels contain adequate knowledge about the process.

5.3. Creating a New View of Processes

One of the main concerns of this research is to create a definitional model for processes which attempts to overcome some of the problems associated with the traditional approach. Let us now consider the critical elements of how we should view processes. The following sections define major attributes of a process model which allows for a comprehensive abstraction capability. Every process may be considered from several different abstraction levels. We can view a single process as a connected set of process nets. The highest net represents the highest amount of process

abstraction while the lowest net in the hierarchy represents the lowest amount of process abstraction.

5.4. Definition of the formal process model

Let us now define a formal model for processes which emphasizes levels of abstraction. We define a process as follows:

$$\textbf{Process}_i = \{ A,K,C \}$$

Process$_i$ is any given process such as "Prepare for donning EMU" or "Replace Battery." Each process is defined as a set of three items: 1) A = Process Abstraction Net, 2) K = Knowledge Base, and 3) C = Control Mechanism. Let us define each of these three items separately.

5.5. Process Abstraction Net

The process abstraction net is the key to the notion of process abstraction since it is here that we define what it means to be a process. We define A (the process abstraction net) to be a hierarchy of separate machines with connecting links provided to traverse the hierarchy. To successfully define a process, we need first use a specific computational model.

5.5.1. Choosing a formal model

Given the criteria in the previous chapter, we must choose a model of computation which satisfies those criteria. We briefly discuss four models and discuss their merits. Finite state automata are a simple and effective means of modeling processes. The primary disadvantage of finite state machines is that they do not model concurrency. Turing Machines are more general and powerful and can model concurrency. Turing machines, while being good at modeling items such as computational complexity, lack an intuitive representation for parallelism and asynchronicity. First order logic can also be used as the basis for modeling processes, however logic seems more appropriate when modeling relationships rather than specific concurrent actions.

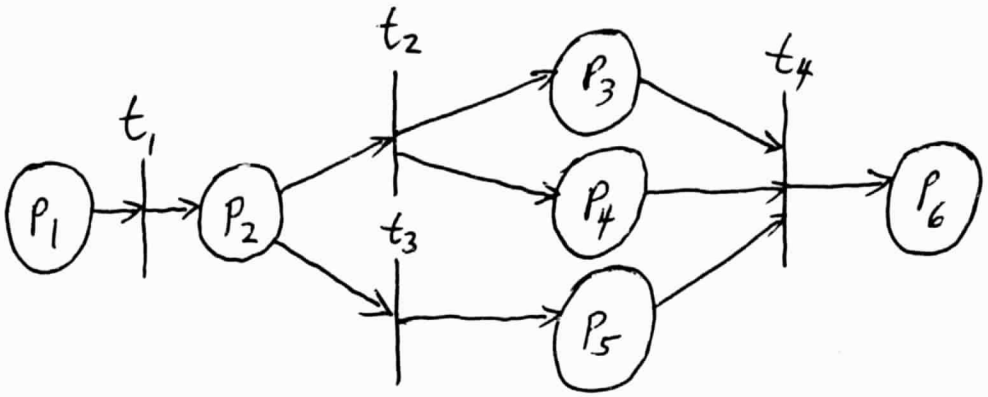
We decide to use Petri Nets [Petri 62] to model each abstraction level. There are

several reasons for the use of Petri Nets. Petri Nets are ideal for modeling asynchronous, concurrent systems. Much of their appeal lies in the visual representation of asynchonicity and concurrency: one may easily view a Petri Net and note the flow of control afforded to the net. They have wide application in many areas and their potential for both modeling and analysis is promising. Before describing Petri Nets, we should note that Petri created the basic notions in his thesis, but a considerable amount of work has been performed since that time: Petri Nets, much like Turing Machines, have been extended to allow for greater and more complete modeling power. A general survey of Petri Net Theory may be found in [Peterson 77] and a comprehensive book on the subject is [Peterson 81].

It is assumed that the reader has a rough knowledge of Petri Nets. A brief description is given in any case. A Petri Net is defined as a four-tuple *(P,T,I,O)*, where P = a set of places or conditions, T = a set of transitions, I = a function from transitions to a bag of places, and O = a function from transitions to a bag of places. Figure 5-1 gives an example net.We may see various forms of parallelism and mutual exclusion in the this net. For instance, we see that places p3 and p4 will occur in parallel. Also, transitions t2 and t3 have the property of being mutually exclusive events if we view the net as a marked net with a single marker in p2.

One may also model conditional execution by specifying the conditions as places in the net which are tied to transitions whose firing depends on the condition.

Our Petri Net Model satisfies our basic criteria for modeling a single process level within the abstraction net *A*. This model serves as a good formal model, however, a more informal model directly derived from this Petri net model is more adequate when thinking in terms of how we might build an actual implementation. Specifically, we choose the production system paradigm since it has wide application and represents the features that we have discussed.




```
P = { p1,p2,p3,p3,p4,p5,p6 }
T = { t1,t2,t3,t4 }

I(t1) = {p1}                O(t1) = {p2}
I(t2) = {p2}                O(t2) = {p3,p4}
I(t3) = {p2}                O(t3) = {p5}
I(t4) = {p3,p4,p5}          O(t4) = {p6}
```

Figure 5-2: An example Petri Net

5.5.2. Choosing an informal model

Given the Petri net model in the previous section, we might ask ourselves "How do we execute the net?" Clearly, the formal model is best at giving us a method for visually portraying information flow. It is not immediately apparent how we should build a real system based on this model. There is a need for specifying a more informal model which can be used as a basis for implementation.

A good approximation to the Petri net formalism is the use of production systems [Post 43]. Production systems have been widely used in various implementations and represent a simple control mechanism. This simplicity of control can be both a benefit and burden to our efforts. The benefit is that we may easily execute production system rules using forward or backward chaining methods. Many

control paradigms are limited to forward chaining and while it is true that forward chaining is our primary concern in simulation, the backward chaining capability is useful for deriving preconditions given a goal. This latter capability is most often exploited in planning systems. The disadvantage to using production systems is the other side of the coin of simplicity. Overall methods for control are somewhat obscured by the restrictions imposed by production systems.

We define $A = \{ P_1, P_2, P_3, ..., P_n \}$ where the individual P_i may be viewed either as Petri Nets or as an equivalent set of production rules. Each transition in the formal Petri net model corresponds to a rule in a production system. Each rule in a production system requires a set of preconditions before "firing" much like a Petri net. Once the rule is fired, a set of postconditions takes effect. P_i is defined as a set of production rules $\{R_1, R_2, ..., R_m\}$. Each R_i is defined as a rule frame. The frame is composed of slot names and values:

Slot Name	Type	Example Value
idnum	decimal	23
idsym	list	(EVA)
def	string	"Extra Vehicular Activity"
preconds	list	((PRE-EVA))
delete	list	((PRE-EVA))
implicit	list	((IN-EVA)(IN-MMU))
explicit	list	(P_2 (DO-EVA))

These slot names and values deserve brief descriptions:

- *idnum* - The identification number for the rule. In the above case, this is rule number 23 in the given production rule set.
- *idsym* - This identifies the actual event taking place. In this case we are defining the rule for performing an EVA.
- *def* - An English description of the event.
- *preconds* - A list of preconditions for the rule to fire.
- *delete* - A list of assertions to delete once the rule has fired (i.e. once the preconditions have been satisfied).
- *implicit* - A list of implicit assertions to be added to the knowledge base when the rule fires.
- *explicit* - A list composed of two elements: the first element is the

identification of a new production rule set to use, and the second element is an assertion used to start execution within this new rule set.

Most of the above slot specifications are quite common in production systems. The exceptions are the last two slots: ***implicit*** and ***explicit***. These are critical in our design of process abstraction, so we discuss these at some length in the next two sections.

5.5.3. Implicit Semantics

We define implicit semantics to be the qualitative attributes assigned to either a place or transition. Implicit semantics are by nature subjective since there definition depends on the knowledge of the particular individual running a simulation based on the given process hierarchy. Within the context of production systems, we choose to assign implicit semantics to the transitions (i.e. production rules) and not to the places (i.e. preconditions). The implicit semantics of a transition are those ***inferred*** by an individual when receiving only the information contained within the identification of the transition . It is easiest to imagine this inference of information as being based on simple message passing between humans: If someone gave a message to another which said "John is performing an EVA" what implicit information could be inferred? As mentioned before, these inferences are rather dependent on the individuals involved: a formal protocol performed with many subjects might determine the greatest overlap in such inferred information.

If we reference the transition EVA, for instance, in the level 2 machine, we can infer several pieces of knowledge:

- We can create a volume which defines the new location of the operator. This volume can be defined possibly as a sphere of a certain radius minus the volume specified by the shuttle. The idea is that given the message EVA, we can certainly identify the agent's new "location volume." The agent is certainly not inside the ship during an EVA and also the agent will not be too far from the shuttle. Hence, we use the notion of a bounded volume to qualitatively describe the agent's new location during the EVA.

- The agent will be wearing an EMU and will be operating the MMU (manned maneuvering unit).

- Probabilistic knowledge such as the estimated position of the agent. We

are certain of the already-defined location volume. This volume defines the boundaries associated with location. It is also possible that we might assign belief measures to various discrete points within the volume.

We can certainly imagine how we can simulate the EVA process completely at process abstraction level 2 by simply executing the net at that level and using only implicit knowledge.

5.5.4. Explicit Semantics

Explicit semantics for a place or transition in a net is a more common notion in computer science. We simply create a subnet at the next lower level of abstraction and attach it to the given place or transition. The explicit semantics of a transition, for instance, is simply defined by the subnet attached to the transition.

5.6. Knowledge Base

The second element of a given $Process_i$ is the knowledge base K (recall that $Process_i = \{ A,K,C \}$). We define the knowledge base to be similar to those defined in standard production systems: a list of assertions. The only item that we change is that we define K to be a list of elements where each element is a list composed of two other elements: an identifier specifying the level of process abstraction and the assertion made at that level.

5.7. Control Mechanism

The control mechanism that we use is essentially the same control mechanism used in standard production systems. The primary difference lies in a choice which is made at each rule firing: should we execute implicit or explict semantics? In the simplest case, we can make this a manual choice. That is, we can step through a simulation of a process and manually choose whether to either stay on the same abstraction level or to traverse the hierarchy in an upwards or downwards fashion.

5.8. Simulation with Process Abstraction

Now that we have defined the components of process abstraction, we can see how we might create a simulation system by just using a process abstraction net. We can allow a user to traverse the many levels of process abstraction by moving down the connecting links between each level. One could either make moves down a level, thereby exploiting the explicit process knowledge at the next level, or execute the Petri Net at the current level, thereby exploiting implicit process knowledge.

This hierarchical view of process abstraction is important, but we should now consider object abstraction and finally spatial abstraction. It is only then, can we form a complete theory which unifies the different types of abstraction.

Chapter 6
Object Abstraction

This chapter is currently under development.

Chapter 7
An Initial Implementation

7.1. Overview

This chapter represents an initial investigation into hierarchical reasoning in terms of an implementation performed in the February 1985 timeframe. The implementation is based on a system called ROSS [Klahr 80] which is an lisp-based, object-oriented package. A more comprehensive implementation is proposed in the future which is predicated on the later-defined concepts defined in the chapters on process and object abstraction. This more comprehensive package will probably be more closely tied to the production system definition already described.

7.2. Abstraction Functions

HIRES supports a set of functions that are useful in providing the analyst with the ability to create a hierarchical reasoning system. The functions fall into two categories: functions that the simulation designer uses to create the actual simulation and functions that allow the simulation user to control the flow of the simulation. This section deals with the *designer functions*.

7.2.1. Process Abstraction

The phrase "current process abstraction level set by the user" will appear in the following definitions. These user defined levels are created with the *flow functions* described in the following section.

```
(0 <level> <object> <message>)
```

Send a <message> to <object> only if <level> <= the current process abstraction level set by the user.

```
(@> <level> <object> <message> <time>)
```

Schedule a <message> to be sent to <object> at <time> only if <level> <= the current process abstraction level set by the user.

```
(@< <level> <object> <message>)
```

De-schedule all <message>'s to be sent to <object> only if <level> <= the current process abstraction level set by the user.

7.2.2. Object Abstraction

```
(# <level-list1> <func1> <level-list2> <func2> ...)
```

<level-list>'s are lists of object abstraction levels. <func>'s are functions that are evaluated only if one of the members of the current object abstraction level list specified by the user is a member of the corresponding <level-list>.

7.2.3. Report Abstraction

```
(% <func1> <func2> ...)
```

If the process abstraction level currently active within the simulation is within the list of report abstraction levels specified by the user, then evaluate the arguments of % (namely, <func1>, <func2>, etc.). These functions will most often be text and/or graphics output functions.

7.3. Controlling the Flow of Simulation

The previous section on abstraction functions specified levels of abstraction that were created by the simulation designer. This section describes some of the *flow functions* which permit the simulation user to "tune" the simulation while it is running. This "tuning" is similar to tuning a television set or radio: one wishes to filter out some aspects while highlighting others.

7.3.1. Starting the Simulation

First, the user must place one or more events onto the event queue, or send one or more messages to objects. Abstraction variables should also be set (see the next section). Then the simulation can begin by telling the simulation clock to tick for a given amount of time. At any time, the resolution of the clock may be changed.

7.3.2. Assigning Abstraction Levels

This section includes two terms which should be defined before continuing: levels and and-or trees. A "<level>" is used to identify an abstraction level. Theoretically, there are infinitely many abstraction levels that can be represented, however, it is pragmatic and efficient to represent only a discrete number of levels. High abstraction levels are represented as low numbers, while low abstraction levels are represented as high numbers. This may sound unduly complicated, although this mapping is appropriate since our simulation is a top-down design: we are more likely to refine our higher levels into still lower levels (and this will simply necessitate increasing the level # as we proceed). It is important to note that when one specifies a procedural abstraction level, all levels less than or equal to that level are effected. This is due to the top-down nature of creating the simulation, and the requirement of maintaining consistency while traversing abstraction levels. Procedural abstraction levels are not independent of one another - they are related in a hierarchical fashion.

The term "<and-or>" tree is also used. This is a goal-tree with *and* and *or* nodes. The value of the tree is true or false, depending on the evaluation at each node. Within the and-or tree, we have items of the form (<predicate> <arg1> <arg2> ...). Some typical values for <predicate> are *class*,*object*,*message*, and *time*.

The interactive functions are now described:

```
(@= <level1> <and-or1> <level2> <and-or2> ...)
```

This is how one tunes the process abstraction in the simulation. Whenever the "@" is encountered during the simulation, the <and-or>'s are evaluated until a true one is found. If the current process abstraction level is less than or equal to the corresponding <level> of a true <and-or> evaluation, the corresponding code (in the "@" call) is evaluated. The "@>" and "@<" functions operate similarly.

An example will demonstrate how @= is used. Suppose that we take the previous example of an agent entering and exiting a room. Let us further suppose that we are interested in carefully monitoring instances when the agent picks up any object within the time range of 500 <-> 800 (in terms of the simulation clock). Any other action is to be simulated at a high level of abstraction. At the beginning of the simulation, we could specify:

```
(@=  3 (and (message (pick up >object))
            (object agent)
            (time 500 800))
     1 true)
```

We are assuming that 3 represents the lowest-level of abstraction defined for this simulation. If the first condition (i.e. <and-or>) is not evaluated as true, then the default will be level 1.

```
(#= <level-list1> <and-or1> <level-list2> <and-or2> ...)
```

Whenever the "#" is encountered during the simulation, the <and-or>'s are evaluated until a true one is found. If the corresponding <level-list> of a true <and-or> evaluation intersects with the <level-list>'s in the "#" function, the corresponding code (in the "#" call) is evaluated.

Object abstraction may be related to objects in the same manner as with the @= function. The main difference is that with object abstraction, we can specify level lists and not just single levels. Using a list of levels means that we can optionally run the simulation while simultaneously viewing several object abstraction level outputs. For instance, text is often output at high level object abstractions while graphics is output at lower levels - the text and graphics could be simultaneously viewed by creating dedicated viewports on the screen. If, in our room example, we wanted to monitor 1) all switches inside room "A" and 2) any agents inside room "B" using a high level of abstraction, we could do the following (assume all other objects/situations are to be modeled using object abstraction level 2).

```
(#= (1) (or (and (class switch-type)
                 (inside-room A))
            (and (class agent-type)
                 (inside-room B)))
    (2) true)
```

Finally, we can specify report abstraction levels to be monitored using this method:

```
(%= <level-list1> <and-or1> <level-list2> <and-or2> ...)
```

The arguments are the same as specified for the function "#=". Whenever the "%" is encountered during the simulation, the <and-or>'s are evaluated until a true one is found. If the current process abstraction level is an element of the corresponding <level-list> for a true evaluation, then the arguments of the "%" function are evaluated. If we always wanted only a high-level of reporting while running a very low-level simulation, we could do the following:

```
(@= 3 true)
(%= (1) true)
```

7.4. An Example: A Day in the Life of a Clock

ROSS will be used for the initial HIRES implementation, due to its capabilities and additional simulation features (such as an event scheduling queue). An example of the use of the HIRES functions will be described in this section. The example is a fairly simple one: the operation of an analog alarm clock. This example was chosen since it is short and concise, yet contains many of the concepts delineated so far. Namely, one can see causality, abstraction levels (for processes,objects, and reports), and interrupts (the alarm feature).

Figure 7-1 gives an overall procedural abstraction hierarchy for the daily activity of the clock:

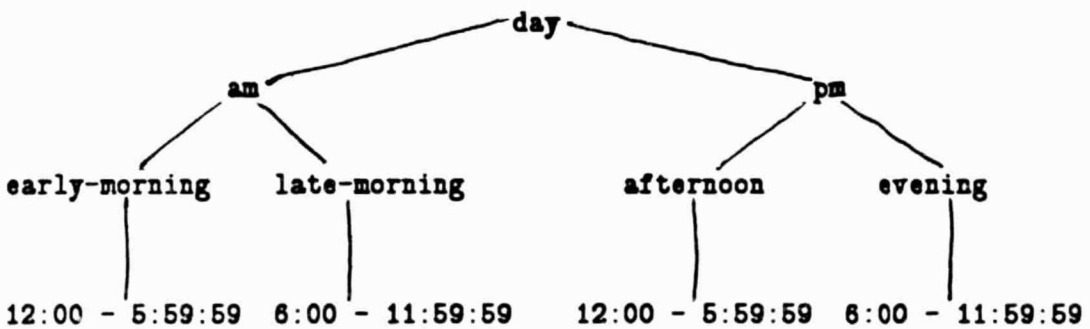


Figure 7-1: Alarm Clock Hierarchy

The HIRES implementation is shown in appendix A. The implementation consists primarily of the four objects (clock, hour-hand, minute-hand, and second-hand) and their associated frames. The *recv* slot name represents message patterns that may cause the corresponding expressions to be evaluated.

There are a few support functions listed in the implementation: (update-graphics <object>) updates a graphics display for <object>; (show-time) displays the current time in a digital format; (output) is used to output data to the terminal device; and (update-coords) keeps track of the two-dimensional coordinate positions for each hand.

Some characteristics of the analog clock in terms of its component parts are now discussed:

- *Clock* - The object "clock" is used to store the current clock time (which is independent of the simulation time), and various methods that can be found within the "recv" slot. The alarm is set by scheduling the pulling of an alarm button. Pushing the button stops the alarm. The "sound alarm" method informs the user that the alarm is continuing to ring if the user has specified a low-level of object abstraction for this instance (at a higher level, we may not be concerned about this detail).

- *Hour-hand* - The "hour-hand" object is designed to move from one tick position to the next. There are four tick positions between each hour on the clock face. The objects "minute-hand" and "second-hand" are similar in operation. Note the two levels of object abstraction for each object: At the high level we are interested in knowing the time in a text format while at the low level we are interested in a more precise definition through a graphics rendition.

A sample interactive HIRES session is presented.

```
% hires
HIRES V1.0 - Hierarchical Reasoning System

; first, let's set up the environment

> (setq $time 0)
> (load 'clock)
Ok..environment "clock" has been loaded

; set abstraction variables to tune the simulation

> (@= 2 true)  ; process abstraction <= level 2 all the time
> (#= (1) true)  ; object abstraction = level 1 all the time
> (%= (1 2) true)  ; report all process levels (all the time)

; next, we schedule an event to get things started

> (@> 1 clock (day) $time)
> (go)  ; start the actual simulation

It is AM
It is PM
A day has passed!

; Let's say that we are interested in the minute-hand during
; the first ten minutes after the clock strikes midnight. Let's
; further assume that we do not want details of the actual
; minute hand (i.e. text output will do). In addition, we will
; set the alarm.

> (setq $time 0)
> (@= 5 true)  ; process abstraction <= level 5 all the time
> (#= (1) true)  ; object abstraction = level 1 all the time
> ; report at process level 5 only when the current object =
> ; "minute-hand" and the current simulation time is within
> ; the specified range (first 10 minutes). Otherwise, do not
> ; report anything
> (%= (5) (and (object minute-hand) (time 0 600)) () true)
> (@> 1 clock (day) $time)
> (@ 1 clock (set alarm 0 1 0))  ; set the alarm
The alarm has been set at 12:01 am
> (@> 1 clock (push button) 342)  ; schedule an interrupt
> (go)

The time is 12:00:00 am
The time is 12:01:00 am
The time is 12:02:00 am
The time is 12:03:00 am
The time is 12:04:00 am
The alarm has been turned off at 12:05:00 am
```

```
The time is 12:05:00 am
          .
          .

The time is 12:10:00 am

> (exit)
Bye
```

Chapter 8
Conclusions

The goal of hierarchical reasoning is to provide a flexible method of simulation by taking advantage of the inherent abstraction levels associated with processes, objects, and their interactions. The analyst may create a knowledge base with many different knowledge layers, and then reason about the change in the environment over arbitrary time intervals. Initial goals have been met by providing a rigorous model for process and object abstraction and by creating an initial implementation for testing purposes.

This thesis has demonstrated the importance of reasoning in a hierarchical fashion, and not simply through strictly qualitative or quantitative methods. It is hoped that both the scientific/engineering community and the artificial intelligence community can benefit by using an integrated, hierarchical approach similar to the one defined. It is critical that all levels of abstract knowledge be available to facilitate a true process simulation.

There remains much work to be done in the area of hierarchical reasoning. The abstraction formalisms presented in the thesis will aid both the designer and user of large simulations by allowing them to reason about portions of a complex environment at different levels.

Chapter 9
References

[Ahuja 80] Ahuja, N., Chien, R. T., and Bridwell N.
Interference Detection and Collision Avoidance among Three Dimensional Objects.
In *Proceedings of the first Annual National Conference on Artificial Intelligence*, pages 44 - 48. AAAI, August, 1980.

[Allen 81] Allen, James F.
An Interval-Based Representation of Temporal Knowledge.
In *Seventh International Joint Conference on Artificial Intelligence*, pages 221 - 226. IJCAI, August, 1981.

[Allen 83] Allen, James F. and Koomen, Johannes A.
Planning Using a Temporal World Model.
In *Eighth International Joint Conference on Artificial Intelligence*, pages 741 - 747. IJCAI, August, 1983.

[Armstrong 83] Armstrong, Charles H., and Veach, William R.
Flight Data File EVA Checklist.
Technical Report, Mission Operations Directorate-Operations Division, NASA Johnson Space Center, October, 1983.

[Badler 75] Badler, Norman I.
Temporal Scene Analysis: Conceptual Descriptions of Object Movements.
PhD thesis, University of Toronto, February, 1975.

[Badler 79] Badler, Norman I. and Smoliar, Stephen W.
Digital Representations of Human Movement.
ACM Computing Surveys 11:19 - 38, March, 1979.

[Badler 83a] Badler, Norman I., Webber, Bonnie L., Korein, James U. and Korein, Jonathan.
TEMPUS: A System For The Design And Simulation Of Mobile Agents In A Workstation And Task Environment.
In *Proceedings of IEEE Trends and Applications Conference.* March, 1983.

[Badler 83b] Badler, Norman I.
Design of a Human Movement Representation Incorporating Dynamics.
August, 1983.

[Bloom 83] Bloom, Douglas, A.
A User-Oriented Interface Control of an Interactive Computer Graphics System.
Master's thesis, University of Pennsylvania, 1983.

[Bonner 82] Bonner, Susan and Shin Kang, G.
A Comparative Study of Robot Languages.
IEEE Computer 15(12):82 - 96, December, 1982.

[Brooks 83a] Brooks, Rodney A. and Lozano-Perez Tomas.
A Subdivision Algorithm in Configuration Space for Findpath with Rotation.
In *Eighth International Joint Conference on Artificial Intelligence*, pages 799 - 806. IJCAI, August, 1983.

[Brooks 83b] Brooks, Rodney A.
Planning Collision-Free Motions for Pick-and-Place Operations.
The International Journal of Robotics Research 2(4):19 - 80, 1983.

[Calvert 80] Calvert, T. W., Chapman, J., and Patla, A.
The Integration of Subjective and Objective Data in the Animation of Human Movement.
In *Proceedings of the SIGRAPH 80 Conference*, pages 198 - 203. SIGRAPH/ACM, July, 1980.

[CORE 79] Graphics Standards Planning Committee.
General Methodology and the Proposed CORE System.
In *Computer Graphics - A Quarterly Report*. SIGRAPH/ACM, August, 1979.

[CSMP 76] *IBM 1130 Continuous System Modeling Program, Program Description and Operations Manual*
SH20-0905 edition, IBM Program Information Department, Hawthorne, N.Y., 1976.

[de Kleer 75] de Kleer, Johan.
Qualitative and quantitative knowledge in classical mechanics.
Technical Report AI-TR-352, Massachusetts Institute of Technology, December, 1975.

[de Kleer 77] de Kleer, Johan.
Multiple Representations of Knowledge in a Mechanical Problem-Solver.
In *Fifth International Joint Conference on Artificial Intelligence*, pages 299 - 304. IJCAI, August, 1977.

[de Kleer 79] de Kleer, Johan.
Causal and Teleological Reasoning in Circuit Recognition.
PhD thesis, Massachusetts Institute of Technology, January, 1979.

[Dijkstra 68] Dijkstra, E. W.
Cooperating sequential processes.
In F. Genuys(Editor) (editor), *Programming Languages*. Academic Press, 1968.

[Fikes 71] Fikes, R. E. and Nilsson, N. J.
STRIPS: A new approach to the application of theorem proving to problem solving.
Artificial Intelligence 2:189 - 208, 1971.

[Forbus 81a] Forbus, Kenneth D.
A Study of Qualitative and Geometric Knowledge in Reasoning about Motion.
Technical Report AI-TR-615, Massachusetts Institute of Technology, February, 1981.

[Forbus 81b] Forbus, Kenneth D.
Qualitative Reasoning about Physical Processes.
In *Seventh International Joint Conference on Artificial Intelligence*, pages 326 - 330. IJCAI, August, 1981.

[Forbus 83] Forbus, Kenneth D.
Measurement Interpretation in Qualitative Process Theory.
In *Eighth International Joint Conference on Artificial Intelligence*, pages 315 - 320. IJCAI, August, 1983.

[Gentner 83] Gentner, Dedre and Stevens, Albert.
Mental Models.
Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.

[Goldin 81] Goldin, Sarah E. and Klahr, Philip.
Learning and Abstraction in Simulation.
In *Seventh International Joint Conference on Artificial Intelligence*, pages 212 - 214. IJCAI, August, 1981.

[GPSS 67] IBM Corporation.
General Purpose Simulation System 1360 Introductory User's Manual, Report # H20-0304
1967.

[Hendrix 73] Hendrix, Gary G.
Modeling Simultaneous Actions and Continuous Processes.
Artificial Intelligence 4:145 - 180, 1973.

[Hoare 78] Hoare, C. A. R.
Communicating sequential processes.
Communications of the ACM 21(8):666 - 677, August, 1978.

[Hollan 84] Hollan, James D., Hutchins, Edwin L. and Weitzman, Louis.
STEAMER: An Interactive Inspectable Simulation-Based Training System.
AI Magazine 5(2):15 - 27, Summer, 1984.

[Hutchinson 70] Hutchinson, A.
Labanotation.
Theatre Arts Books, 1970.

[Joels 82] Joels, Kerry M. and Kennedy, Gregory P.
The Space Shuttle Operator's Manual.
Random House, Inc., 1982.

[Klahr 80] Klahr, Philip and Faught, William S.
Knowledge-Based Simulation.
In *Proceedings of the first Annual National Conference on Artificial Intelligence*, pages 181 - 183. AAAI, August, 1980.

[Konolige 80] Konolige, Kurt and Nilsson, Nils J.
Multiple-Agent Planning Systems.
In *Proceedings of the first Annual National Conference on Artificial Intelligence*, pages 138 - 142. AAAI, August, 1980.

[Korein 84] Korein, James U.
A Geometric Investigation of Reach.
PhD thesis, University of Pennsylvania, 1984.

[Kuipers 76] Kuipers, Benjamin.
Spatial Knowledge.
Technical Report, Massachusetts Institute of Technology, June, 1976.
AI Memo 359.

[Lebling 79] Lebling, David P. and Blank, Marc S. and Anderson, Timothy A.
ZORK: A Computerized Fantasy Simulation Game.
IEEE Computer , April, 1979.

[Lieberman 77] Lieberman, L. I. and Wesley, M. A.
AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly.
IBM Journal of Research and Development 21(4):321 - 333, July, 1977.

[Lowrance 77] Lowrance, John, D. and Friedman, Daniel P.
Hendrix's Model for Simultaneous Actions and Continuous Processes: An introduction and Implementation.
International Journal of Man-Machine Studies 9:537 - 581, 1977.

[Lozano 79] Lozano-Perez, Tomas and Wesley, Michael A.
An Algorithm for Planning Collision -Free Paths Among Polyhedral Obstacles.
Communications of the ACM 22:560 - 570, October, 1979.

[Mujtaba 82] Mujtaba, M. S. and Goldman, R. and Binford T.
Stanford's AL Robot Programming Language.
Computers in Mechanical Engineering :50 - 57, August, 1982.

[Nilsson 80] Nilsson, Nils J.
Principles of Artificial Intelligence.
Tioga Publishing Company, 1980.

[Payne 82] Payne, James A.
Introduction to Simulation: Programming Techniques and Methods of Analysis.
McGraw-Hill, 1982.

[Peterson 77] Peterson, James L.
Petri Nets.
Computing Surveys 9(3):223 - 252, September, 1977.

[Peterson 81] Peterson, James L.
Petri Net Theory and the Modeling of Systems.
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[Petri 62] Petri, C.
Kommunikation mit Automaten.
PhD thesis, University of Bonn, 1962.

[Post 43] Post, E.
Formal reductions of the general combinatorial problem.
American Journal of Mathematics 65:197 - 268, 1943.

[Pritsker 74] Pritsker, A. A. B.
The GASP IV Simulation Language.
Wiley, 1974.

[QR 84]
Artificial Intelligence.
Volume 24: Numbers 1-3.
December, 1984
Special Volume on Qualitative Reasoning about Physical Systems.

[Rieger 77] Rieger, Chuck and Grinberg, Milt.
The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms.
In *Fifth International Joint Conference on Artificial Intelligence*, pages 250 - 256. IJCAI, August, 1977.

[Sacerdoti 77] Sacerdoti,Earl D.
A Structure for Plans and Behavior.
Elsevier North Holland Inc., 1977.

[Salter 80] Salter, Richard M.,Brennan, Terence J., and Friedman, Daniel P.
CONCUR: A language for continuous concurrent processes.
Computer Languages 5:163 - 189, 1980.

[Salter 83] Salter, Richard M.
Planning in a Continuous Domain-An Introduction.
Robotica 1:85 - 93, 1983.

[Salter 84] Salter, Richard M.
Declarative Modeling of Symbolic Continuous Processes in a Robot Simulation System.
1984.
To appear in *Robotica*.

[Shapiro 84] Shapiro, Lynne A.
A Method for Generating Shadows With Umbra and Penumbra in Computer Generated Images.
Master's thesis, University of Pennsylvania, May, 1984.

[SIMSCRIPT 72] Consolidated Analysis Centers, Inc.
SIMSCRIPT II.5 Reference Handbook
1972.

[Smoliar 77] Smoliar, S. W. and Weber, L.
Using the computer for a semantic representation of Labanotation.
Computing in the humanities :253 - 261, 1977.

[STARCROSS 82]
StarCross User's Manual
Infocom, Inc., 1982.

[Taylor 82] Taylor, R. H., Summers, P. D. and Meyer, J. M.
AML : A Manufacturing Language.
The International Journal of Robotics Research 1(3), 1982.

[Tsotsos 80] Tsotsos, John K.
A Framework for Visual Motion Understanding.
Technical Report CSRG-114, University of Toronto, June, 1980.

[Vere 83] Vere, Steven A.
Planning In Time: Windows And Durations For Activities And Goals.
IEEE Transactions On Pattern Analysis And Machine Intelligence 5(3):246-267, May, 1983.

[Weber 78] Weber, L. ,Smoliar, S.W. and Badler, N.I.
An Architecture for the Simulation of Human Movement.
Proceedings of the 1978 Annual Conference. ACM 1, December 4 - 6, 1978.

[Weinstein 83] Weinsten, Luke A.
A Menu Driven User Interface.
Master's thesis, University of Pennsylvania, 1983.

[Zeltzer 82] Zeltzer, David.
Motor Control Techniques for Figure Animation.
IEEE Computer Graphics and Applications 2:53 - 59, November, 1982.

[Zeltzer 83] Zeltzer, David.
Knowledge-based Animation.
In *SIGRAPH/SIGART Interdisciplinary Workshop : MOTION: Representation and Perception*, pages 187 - 192. Assocation for Computing Machinery, April, 1983.

Appendix A
An Alarm Clock Simulation

```
;==============================
;
; An Alarm Clock Simulation
; written using HIRES capability
;
; Paul Fishwick
;
;==============================
; Specify the actors for the clock simulation
;
; Define the object hierarchy
;
(ask something create instance clock with
     hour 0
     minute 0
     second 0)
(ask something create instance hour-hand with
     x1-coord 0
     y1-coord 0
     x2-coord 0
     y2-coord 5
     color blue)
(ask something create instance minute-hand with
     x1-coord 0
     y1-coord 0
     x2-coord 0
     y2-coord 10
     color green)
(ask something create instance second-hand with
     x1-coord 0
     y1-coord 0
     x2-coord 0
     y2-coord 8
     color red)
;
;  Define the Object Behaviors
;
[ask clock when receiving
 (day)
     (@ 2 clock (am))
     (@> 2 clock (pm) 43200) ; pm starts in 12 hours
     (@> 1 clock (next day) 86400)]

[ask clock when receiving
  (next day)
     (# (1 2) (% (output "A day has passed!")))]

[ask clock when receiving
  (am)
     (# (1 2) (% (output "It is AM")))
     (@ 3 clock (early morning))
     (@> 3 clock (late morning) 21600)] ; late morning in 6 hours

[ask clock when receiving
  (pm)
     (# (1 2) (% (output "It is PM")))
     (@ 3 clock (afternoon))
     (@> 3 clock (night) 21600)] ; night is 6 hours from now

[ask clock when receiving
```

```
  (early morning)
     (# (1 2) (% (output "It is in the early morning")))
     (@ 4 hour-hand (move 0 5))]

[ask clock when receiving
  (late morning)
     (# (1 2) (% (output "It is in the late morning")))
     (@ 4 hour-hand (move 6 11))]

[ask clock when receiving
  (afternoon)
     (# (1 2) (% (output "It is the afternoon")))
     (@ 4 hour-hand (move 12 17))]

[ask clock when receiving
  (night)
     (# (1 2) (% (output "It is nighttime")))
     (@ 4 hour-hand (move 18 23))]

[ask clock when receiving
  (set alarm >hours >minutes)
     (setq seconds 0)
     (setq sim-time (plus (times hours 3600) (times minutes 60) seconds))
     (# (1) (% (output "The alarm has been set at "
                          (show-alarm-time hours minutes)))
        (2) (% (update-graphics "alarm-buttom")))
     (@> 1 clock (sound alarm) sim-time)]

[ask clock when receiving
   (sound alarm)
     (# (2) (% (output "The alarm is ringing ...")))
     (@> 1 clock (sound alarm) 1)]

[ask clock when receiving
   (push button)
     (# (1) (% (output "The alarm has been turned off at " (show-time)))
        (2) (% (update-graphics "alarm-buttom")))
     (@< 1 clock (ask clock sound alarm))] ; de-schedule (sound alarm)


[ask hour-hand when receiving
   (move >start >stop)
     (if (le start stop)
         ;(update-coords "hour hand")
         (# (1) (% (output "The time is " (show-time)))
            (2) (% (update-graphics "hour hand")))
         (setq expression (list '@> 4 'hour-hand
             (list 'nexthourtick start stop)  720))
         (eval expression)
         (@ 5 minute-hand (move 0 11))]

[ask hour-hand when receiving
   (nexthourtick >start >stop)
     (set-slot clock hour (plus (get-slot clock hour) 0.2))
     (set-slot clock minute
          (mod (fix (times (get-slot clock hour) 60)) 60))
     (setq expression (list '@ 4 'hour-hand
          (list 'move (plus start 0.2) stop)))
     (eval expression)]
```

```
[ask minute-hand when receiving
   (move >start >stop)
     (if (le start stop)
         ;(update-coords "minute hand")
         (# (1) (% (output "The time is " (show-time)))
            (2)  (% (update-graphics "minute hand")))
         (setq expression (list '@> 5 'minute-hand
             (list 'nextminute start stop) 60))
         (if (ne start stop) (eval expression))
         (@ 6 second-hand (move 0 59))]


[ask minute-hand when receiving
   (nextminute >start >stop)
     (set-slot clock minute (plus (get-slot clock minute) 1))
     (set-slot clock second 0)
     (setq expression (list '@ 5 'minute-hand
          (list 'move (plus start 1) stop)))
     (eval expression)]


[ask second-hand when receiving
   (move >start >stop)
     (if (le start stop)
         ;(update-coords "second hand")
         (# (1) (% (output "The time is " (show-time)))
            (2) (% (update-graphics "second hand")))
         (setq expression (list '@> 6 'second-hand
             (list 'nextsecond start stop) 1))
         (if (ne start stop) (eval expression))]

[ask second-hand when receiving
   (nextsecond >start >stop)
     (set-slot clock second (plus (get-slot clock second) 1))
     (setq expression (list '@ 6 'second-hand
         (list 'move (plus start 1) stop)))
     (eval expression)]


;
;  Define support functions
;
(defun output fexpr (out-list)
   (mapcar '(lambda (a-value) (princ (eval a-value)))
           out-list)
   (terpri))

(defun update-graphics (type)
   (output "Graphics are being updated for the " type))

(defun update-coords (type)
   (output "Coordinate locations are being updated for the " type))

(defun show-time nil
   (setq hour (fix (get-slot clock hour)))
   (setq indicator 'pm)
   (if (le hour 11) (setq indicator 'am))
   ; adjust hour to 12 hour format with am/pm indicator
   (setq hour (plus (mod (+ hour 11) 12) 1))
   (setq minute (fix (get-slot clock minute)))
```

```
   (setq second (get-slot clock second))
   (princ hour) (princ '|:|)
   (princ minute) (princ '|:|)
   (princ second)
   (princ '| |)
   (princ indicator))

(defun show-alarm-time (hour minute)
   (setq indicator 'pm)
   (if (le hour 11) (setq indicator 'am))
   ; adjust hour to 12 hour format with am/pm indicator
   (setq hour (plus (mod (+ hour 11) 12) 1))
   (princ hour) (princ '|:|)
   (princ minute) (princ '| |)
   (princ indicator))

(defun get-slot fexpr (list)
   (setq object (car list))
   (setq slot (cadr list))
   (ask !object recall your !slot))

(defun set-slot fexpr (list)
   (setq object (car list))
   (setq slot-name (cadr list))
   (setq slot-value (eval (caddr list)))
   (ask !object set your !slot-name to !slot-value))
```